INTERACTIVE VISUALIZATION AND EXPLORATION OF FEATURE EVOLUTION IN DYNAMIC DATA

by

Wathsala Widanagamaachchi

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

August 2017

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Wathsala Widanagamaachchi**
has been approved by the following supervisory committee members:

Valerio Pascucci ,	Chair(s)	**22 May 2016** Date Approved
Feifei Li ,	Member	**08 June 2016** Date Approved
Jeffrey Phillips ,	Member	**22 May 2016** Date Approved
Paul Rosen ,	Member	**22 May 2016** Date Approved
Peer-Timo Bremer ,	Member	**22 May 2016** Date Approved

by **Ross T. Whitaker** , Chair/Dean of
the Department/College/School of **Computing**
and by **David B. Kieda** , Dean of The Graduate School.

ABSTRACT

A broad range of applications capture dynamic data at an unprecedented scale. Independent of the application area, finding intuitive ways to understand the dynamic aspects of these increasingly large data sets remains an interesting and, to some extent, unsolved research problem. Generically, dynamic data sets can be described by some, often hierarchical, notion of feature of interest that exists at each moment in time, and those features evolve across time. Consequently, exploring the evolution of these features is considered to be one natural way of studying these data sets. Usually, this process entails the ability to: 1) define and extract features from each time step in the data set; 2) find their correspondences over time; and 3) analyze their evolution across time. However, due to the large data sizes, visualizing the evolution of features in a comprehensible manner and performing interactive changes are challenging. Furthermore, feature evolution details are often unmanageably large and complex, making it difficult to identify the temporal trends in the underlying data. Additionally, many existing approaches develop these components in a specialized and standalone manner, thus failing to address the general task of understanding feature evolution across time.

This dissertation demonstrates that interactive exploration of feature evolution can be achieved in a non-domain-specific manner so that it can be applied across a wide variety of application domains. In particular, a novel generic visualization and analysis environment that couples a multiresolution unified spatiotemporal representation of features with progressive layout and visualization strategies for studying the feature evolution across time is introduced. This flexible framework enables on-the-fly changes to feature definitions, their correspondences, and other arbitrary attributes while providing an interactive view of the resulting feature evolution details. Furthermore, to reduce the visual complexity within the feature evolution details, several subselection-based and localized, per-feature parameter value-based strategies are also enabled. The utility and generality of this framework is demonstrated by using several large-scale dynamic data sets.

To Lalindra for his endless love and support.

CONTENTS

LIST OF TABLES

ACKNOWLEDGEMENTS

This dissertation could not have been accomplished without the help of many whom I would like to thank. First, I would like to thank my family, especially my husband Lalindra, whose endless support and encouragement made this work possible. Second, I would like to thank my advisor and mentor, Valerio Pascucci, for his continued guidance and encouragement. Third, I would like to thank the other members of my committee, Timo, Paul, Jeff, and Feifei, for their feedback on this work. Fourth, I would like to thank Jim along with Chris and Ollie, for their continued guidance throughout my work period at Los Alamos National Laboratory. Finally, I would also like to thank the many labmates and collaborators from the Data Analysis group along with my many friends in Salt Lake City.

PART I

INTRODUCTION AND BACKGROUND

CHAPTER 1

MOTIVATION AND CONTRIBUTIONS

A wide variety of data are being captured in science, engineering, medicine, and business and in many other day-to-day activities. As the available computing power and storage capacities increase, both the resolution and complexity of these data also grow at a staggering rate. For example, one of the largest cosmological simulations ever produced, the Q continuum simulation, models the evolution of the universe from just 50 million years after the Big Bang to the present day, involving half a trillion particles [1]; about 100,000 weather sensors worldwide (with 40,000 in the United States alone) gather various observations of atmospheric, terrestrial, and oceanic conditions in real time [2]; higher resolution weather simulations of Hurricane Sandy, consisting of around a 500-meter resolution, the equivalent of a few city blocks, are conducted by researchers at the National Center for Atmospheric Research and the University of Illinois [3]; worldwide digital healthcare data were estimated to be equal to 500 petabytes in 2012, and are expected to reach 25,000 petabytes in 2020 [4]; and there are 347,222 tweets on Twitter, 701,389 logins on Facebook, 150 million emails sent, 2.4 million search queries on Google, 2.78 million video views on YouTube, and 28,194 new posts to Instagram appearing in a single minute of real time on the Internet [5]. Researchers are interested in finding effective ways to understand this ever-increasing abundance of data.

To this end, understanding the dynamic nature of data is considered to be an important research direction. One common means of achieving this is via exploring the evolution of features across time. Given a dynamic data set, such an analysis usually proceeds in four steps: first, the feature of interest within the data set is defined; second, features for each time step (i.e., a snapshot in time) within the data set are extracted; third, feature correspondences are established across time steps (i.e., tracked); and fourth, the resulting feature evolution details are analyzed. Although this process appears relatively straight-

forward, it presents a number of practical challenges, especially for the existing data sets of interest. It involves coupling the corresponding analysis both within and across time steps, which increases the difficulty exponentially. This process also multiplies the amount of data that must be considered simultaneously, often exceeding the available memory and other resources. Furthermore, the resulting data potentially contain information about thousands of features across hundreds of time steps, making it challenging to present them in a comprehensible manner. Additionally, given the large data sizes, it is often infeasible to make dynamic modifications to parameters that define the features and their correspondences, especially as part of an interactive exploration and simplification. This dissertation addresses some of these challenges by developing a general and flexible analysis environment that allows interactive exploration of feature evolution within dynamic data sets.

1.1 Dynamic Data

Before further discussion, a description of dynamic data in the context of this dissertation is necessary. 'Dynamic data' refers to information that changes over time. Today, many applications (e.g., combustion, cosmology, healthcare, and Twitter) capture such dynamic data on a regular basis, see Figure 1.1. For example, recent numerical combustion simulations model turbulence within flames for various durations of time in order to gain insights about the impact of turbulence on the combustion process [6]; state-of-the-art cosmological simulations model the universe over billions of years, providing scientists the opportunity to study the evolution of the universe in great detail [1]; on a daily basis, the US healthcare system produces hundreds of thousands of patient records detailing a wide range of information from admission times and dates to symptoms and outcomes [7]; and on Twitter, around 6000 tweets on average are produced every second [8].

One natural way of studying these dynamic data sets is via analyzing their dynamic nature across time. As indicated by the example above, dynamic data sets usually contain observations that are captured over time on either real-world or simulated phenomena. More importantly, each element in the data set is associated with a time stamp, and is usually captured in time steps. For example, a set of tweets captured over time without the exact times they were posted would not be a dynamic data set. Having a time stamp

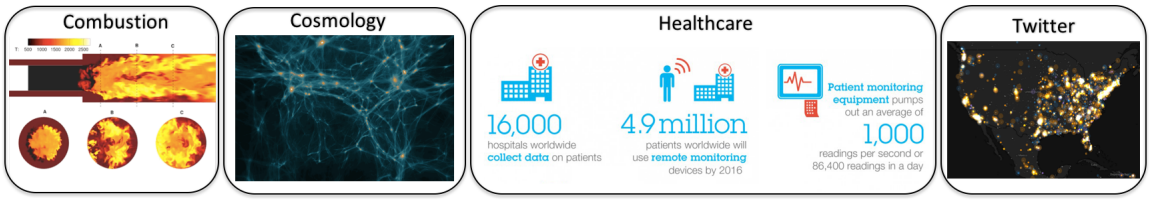


Figure 1.1: Dynamic data are captured in a wide variety of application domains.

associated with every element provides the means to perform the analysis necessary to understand the dynamic nature of the data sets. Additionally, each dynamic data set has some notion of a feature of interest (e.g., burning cells in combustion data, halos in cosmology data, patient groups in healthcare data, and Twitter topics in Twitter data), and these features evolve across time. Exploring and analyzing the behaviors of these features with respect to changes in parameters (e.g., fuel consumption rate in combustion data, distance in cosmology data, patient similarity in healthcare data, and textual similarity in Twitter data) and in time is essential to effectively understand the dynamic aspects of the underlying data.

Furthermore, dynamic data sets can be categorized based on various classifications such as application domain (scientific or nonscientific) and data type (spatial or nonspatial). The separation between science and nonscience is a widely argued topic in the literature [9], [10]. When broadly classified, data resulting from applications in scientific and engineering fall under scientific dynamic data, whereas data from applications in other domains such as social media, healthcare, business, and other day-to-day activities are considered to be nonscientific dynamic data. Additionally, the common types of spatial dynamic data include structured or unstructured grids and point sets, whereas nonspatial dynamic data include text-, graph-, and image-based data.

1.2 Dynamic Data Analysis via Feature Evolution Exploration: Challenges

Over the years, many advances have been made on the topic of feature evolution exploration within dynamic data. However, many important challenges remain for several reasons, including the resolution and complexity of dynamic data sets, limitations in the human capacity for processing information, and limitations of the existing approaches.

1.2.1 Challenges Due to the Large Data Sizes

As our ability to generate and gather data increases, large amounts of dynamic data are captured on a daily basis. The sheer size of dynamic data sets available makes several tasks involved, such as feature evolution computation, comprehensible presentation, and interactive exploration, extremely challenging.

1.2.1.1 Feature Evolution Computation

For a given dynamic data set, computing feature evolution details usually requires coupling data both within and across its time steps. At a particular parameter setting, first, all features from all time steps are extracted, and then, feature correspondences across time are established. The first step itself, extraction of features, is an expensive operation. It typically requires the entire domain, or parts of it, to be inspected for each time step in the data set. Next, to establish feature correspondences across time, all features in each consecutive pair of time steps have to be compared. This step typically requires multiple traversals of the corresponding segmentation as well as construction of suitable search structures. For both steps, there exists a wide range of solutions such as clustering [11], [12] and threshold-based segmentation [13], [14] to define features, and region overlap-based [15], [16] and attribute-based [17–19] techniques for computing feature correspondences.

Nevertheless, even for practical data sets, computing feature evolution entails processing a large number of features for hundreds of time steps. As the resolution and complexity of dynamic data sets increase, the amount of data that need to be processed during this computation also grows exponentially. Accordingly, the memory and time requirements also increase. For instance, for the terabyte-scale data sets common today, computing feature evolution details for a single parameter setting often results in hours or even days of file I/O time alone. To make matters worse, at each change in the parameter settings, features and their correspondences need to be recomputed. Therefore, as the data sizes increase, computing feature evolution details for a specific parameter setting, let alone within an interactive setting, quickly becomes infeasible.

1.2.1.2 Comprehensible Presentation

Depending on the number of time steps within a data set and the amount of features within each of its time steps, the resulting feature evolution details can become large and

complex. Therefore, presenting these details in a comprehensible manner can be challenging. To this end, over the last few decades, visualization has become an increasingly important component of visual analytics used to present feature evolution across time. Traditionally, techniques such as illustration, abstraction, art, morphing, and animation have been used for visualizing the evolution of data [20–22]. As the field of visualization has grown, and depending on the subject area, many different techniques have been developed [23–30]. Among them, one of the most common approaches used to illustrate feature evolution is via a *timeline*, a graphic representation of the passage of time as a line [31], [32].

Nevertheless, regardless of the type of visualization used to illustrate feature evolution, as the data sizes increase, maintaining the comprehensibility of the visualization can be challenging. Consider *tracking graphs*, a timeline-based approach that represents feature evolution as a collection of *feature tracks* that split/merge across time, see Figure 1.2. Compared to other visualizations, these graphs are able to present global overviews of the entire feature evolution in a simple and effective manner. Yet, for larger data sets, even a tracking graph can easily grow beyond the limited number of nodes and edges a human can comprehend. Therefore, computing an optimal layout for such a graph also becomes expensive and involves large amounts of memory and time requirements. Furthermore, for all but the smallest data sets, a tracking graph, even assuming an optimal layout, quickly becomes incomprehensibly large and complex for users to understand, see Figure 1.3.

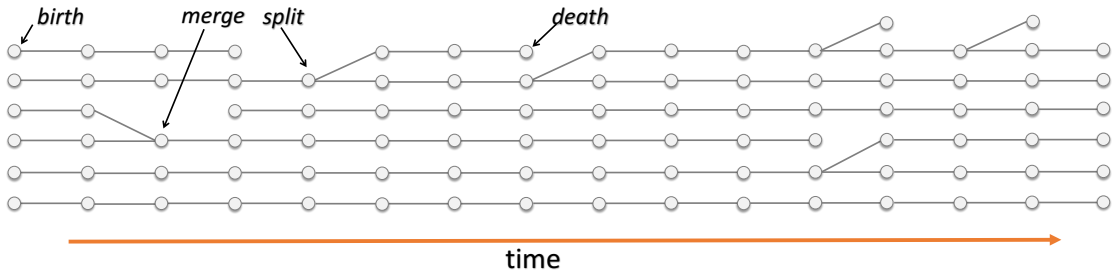


Figure 1.2: A tracking graph showing the evolution of features across time. Each node represents a feature, and its 'track' shows how that feature evolves across time: splitting, merging, or disappearing.

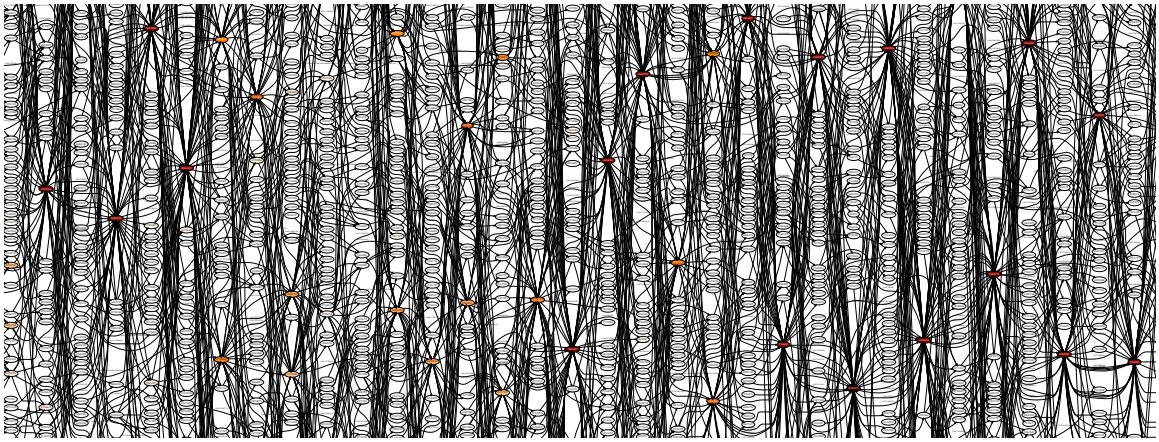


Figure 1.3: A zoomed-in view of a tracking graph for a combustion data set [33], [34] visualized and optimized using Dot [35]. The displayed graph contains only 33 of the 324 time steps and 819 nodes, yet it is still nearly incomprehensible due to the complex interactions.

1.2.1.3 Interactive Exploration and Analysis

Apart from constructing and presenting the feature evolution across time, interactively exploring and analyzing the behaviors of these features is of significant interest. This exploration and analysis includes functions such as exploring the entire parameter space of features, simplifying feature evolution details, and isolating interesting features.

Existing approaches focus on only a few known parameter values within the entire parameter space of features and lack the ability to evaluate those parameter choices in an efficient manner. Therefore, exploring feature evolution across multiple parameter values automatically and effortlessly is of great interest to researchers. It allows one to understand how a certain feature's behavior and attributes change as the parameter values vary, and thus to better characterize that feature. However, as previously discussed, when dealing with large data sizes, feature evolution construction and presentation steps often involve high costs. Consequently, changing feature definitions (by changing the parameter values) within an interactive setting is virtually infeasible.

Furthermore, as data sizes increase, the resulting feature evolution details also become large and complex. Data sets often contain spatially small features (within time steps) that are not necessarily of interest to the analysis. They also contain features that can cause repeated merges and splits across time, thus leading to unnecessary clutter in the resulting

feature evolution details. Due to these reasons, it is often impossible to identify temporal trends in the underlying data, let alone follow a given feature through time. Therefore, it is of interest to simplify feature evolution details based on various feature-based and correspondence-based attributes (e.g., removing spatially small features, eliminating feature correspondences with small weights), and thereby isolate interesting features. Again, due to the costs involved in feature evolution construction and presentation steps, such interactive simplification often becomes challenging.

1.2.2 Challenges Due to the Limitations in the Human Capacity for Processing Information

Today, an estimated 2.5 quintillion bytes of data are generated every single day [36]. In order to make sense of this continuous flood of data, it is essential to translate this information into something meaningful and comprehensible. To this end, data visualization helps to meet this need in a significant way. Humans respond to and process visual data better than any other type of data. In fact, our brain processes images 60,000 times faster than text [37]. Consequently, compared to other techniques, it is more advantageous to utilize data visualization techniques to understand dynamic data sets. In fact, many different techniques have been developed to visualize the temporal evolution of features, including illustration, abstraction, morphing, and animation [20–30].

As the saying goes, "a picture is worth a thousand words" – often more – but only when the information is presented graphically rather than verbally and the visualization is well designed. In the case of dynamic data, especially for the larger data sets of interest, visualizing feature evolution details to best convey the required information can be challenging. Approaches such as morphing and animation, which make use of the human visual system and the brain's memory capacity to convey the feature evolution details, often fail when the data sets contain thousands of time steps. Instead, techniques such as tracking graphs, which present the entire feature evolution in one visual representation, have the potential to be more effective. Even so, a human can perceive only a certain amount of information at a time. Consequently, as feature evolution details become large and complex, it can be difficult to identify trends in the underlying data even with tracking graphs.

To that end, it is essential to leverage existing visualization techniques to effectively present the feature evolution details within dynamic data sets. Furthermore, it is beneficial

to provide the means to highlight or grasp the insights and other important details within the visualization itself. For example, in the case of tracking graphs, instead of just visualizing the feature evolution of the entire data set, it is of interest to have the flexibility to explore the entire data set (e.g., by expanding and narrowing the focus of interest, changing the parameters to change the feature definitions), extract specific feature evolution details, and simplify feature evolution across time within an interactive setting. All these combined have the potential to convey the feature evolution details more effectively, and thus identify underlying patterns and trends within data.

1.2.3 Challenges Due to the Limitations in Existing Approaches

A broad range of application domains capture dynamic data on a daily basis. Irrespective of the application domain, studying the dynamic aspects of these increasingly large data sets is considered to be of significant interest. As previously discussed in Section 1.1, it is interesting to note that dynamic data from different application domains have certain commonalities. Nevertheless, in spite of these commonalities, most existing approaches are customtailored to their specific use cases.

To elaborate, when considering most prior work, methods used for defining and tracking the features of interest vary based on the underlying feature type of the data. For example, techniques such as isosurfaces [38] and interval volumes [39] have been used to identify threshold-based features of a scalar field. Other approaches such as clustering [11], [12], topological analysis [13], [14], region growing [40], [19], and visualization-based techniques [41] have also been used for defining features. For computing feature correspondences across time, again many approaches have been used. When sufficient temporal resolution is present in the source data, region overlap-based methods have been used to establish feature correspondences [15], [16]. Other approaches that compute attribute-based correspondences are also found in the literature [17–19].

Additionally, this problem of exploring feature evolution across time has long been an area of interest within the visualization community. As the field of visualization has grown, very different visualization techniques have been developed to address this problem depending on the subject area, often with a strong separation into scientific visualization (SciVis) and information visualization (InfoVis) categories. In SciVis research, tech-

niques such as illustration, morphing, and animation have been used to visualize feature evolution in scientific data sets [20–22]. In InfoVis, a variety of visualization techniques, such as ThemeRiver [42], StoryFlow [29], EventFlow [43], and OutFlow [44], are used to visualize feature evolution in very different data sets (e.g., social media, financial, video, and source code data). Furthermore, in contrast to the related InfoVis research, the SciVis focus is often on managing data efficiently to produce scalable solutions rather than on the clarity or effectiveness of the visualization.

With respect to the general notion of dynamic features, this specialization aspect observed in the relevant related work in the areas of feature extraction, correspondence computation, and visualization seems rather arbitrary and unnecessary. Instead, an approach for understanding feature evolution across time in a general fashion that is applicable across a variety of application domains, rather than focusing on one specific domain, is more appropriate.

1.3 Dissertation Statement and Contributions

The role of feature evolution exploration in dynamic data analysis is to enable an understanding of spatiotemporal behaviors of features and to identify underlying patterns and trends in data. To effectively and efficiently explore feature evolution within an interactive setting, three important factors should be addressed: extraction, visualization, and simplification of feature evolution across time. Also, in order to ensure such a feature exploration process is applicable across a variety of application domains, a generic design should be maintained across all components involved.

Specifically, the contributions of this dissertation are divided into three main parts following the introductory part: Part II develops a novel multiresolution unified spatiotemporal representation of features to enable interactive extraction of feature evolution across time; with a focus on tracking graphs, Part III develops a novel progressive layout and visualization strategy to enable interactive visualization of feature evolution details, and introduces subselection-based and localized, per-feature parameter value-based strategies for reducing the visual complexity of feature evolution details; and Part IV develops a novel visualization and analysis environment to explore feature evolution across time in a generic manner.

1.3.1 Dissertation Statement

Understanding the dynamic characteristics of features in data can substantially benefit from a non-domain-specific approach to explore feature evolution across time. Current techniques are custom tailored to their specific use cases, and thus fail to address the general task of understanding feature evolution across time. The research done as part of this dissertation has led to a novel generic framework that couples a multiresolution unified spatiotemporal representation of features with progressive layout and visualization strategies for understanding feature evolution across time in a more general fashion. The utility and generality of this framework are demonstrated by using dynamic data sets from a range of application domains.

1.3.2 Dissertation Contributions

The key contributions of this dissertation are itemized as follows:

- **Multiresolution unified spatiotemporal representation of features for fast extraction of feature evolution details** (Part II, Chapters 3, 4, 5)

  As mentioned in Section 1.2.1.1, given the large data sizes, one of the major challenges in interactively exploring feature evolution is making dynamic modifications to parameters that define features and their correspondences. As a result, this dissertation develops a novel multiresolution unified spatiotemporal representation of features that can encode either univariate or bivariate features and their correspondences for their entire parameter range.

  First, a *feature hierarchy* that encodes the clustering hierarchy of *univariate features* (i.e., features defined using a single parameter) for a range of parameter values is presented in **Chapter 3**. For a particular time step, instead of computing a set of univariate features per parameter value, this feature hierarchy constructs a meta-representation that stores all possible features and their feature-based attributes for the entire parameter range. Therefore, once the feature hierarchy is constructed, univariate features and their feature-based attributes can be quickly and easily extracted for any parameter within its range.

  Second, a novel hierarchical feature representation, called *bivariate graph*, that effi-

ciently and compactly encodes the hierarchical relationships between *bivariate features* (i.e., features defined using two parameters) for a range of parameter values is introduced in **Chapter 4**. Specifically, a simple yet effective approach to combine two univariate feature hierarchies into a single bivariate one is presented. Given any value for both parameters, just as with the univariate feature hierarchy, this structure also has the capability to quickly extract bivariate features and their feature-based attributes for the entire parameter range. These two hierarchical feature representations remove the need for repeated feature computation in the case of univariate and bivariate features and make interactive feature extraction possible.

Third, a novel and flexible structure, called *meta graph*, that encodes feature correspondences for a range of parameter values is introduced in **Chapter 5**. Meta graph, similar to the aforementioned feature representations, stores not only feature correspondences for one particular parameter value, but also the entire family of feature correspondences for all possible parameter values. Together, these data representations enable fast extraction of feature evolution details for any particular parameter value within the entire parameter range.

- **Progressive layout and visualization strategy that enables interactive visualization of feature evolution** (Part III, Chapter 6)

  As discussed in Sections 1.2.1.2 and 1.2.2, techniques that convey information about the dynamic nature of data through visual representations are becoming increasingly important and necessary. This dissertation visualizes feature evolution details using tracking graphs with which concise representations of feature evolution are captured as a collection of feature tracks. Tracking graphs have the capability to present global overviews of feature evolution in a simple and effective manner.

  In **Chapter 6**, with a focus on tracking graphs, a novel progressive graph layout and visualization strategy that allows interactive visualization of feature evolution across time is introduced. Specifically, this proposed strategy always processes a tracking graph with respect to a focus time step and a window of interest. Starting from the focus time step, features and their correspondences are iteratively added both forward and backward in time up to the window of interest to form the tracking

graph. Then, a progressive two-stage graph layout algorithm that utilizes a fast initial layout and a slower greedy layout is used to interactively visualize tracking graphs.

- **Subselection-based and localized, per-feature parameter value-based strategies for reducing the visual complexity of feature evolution** (Part III, Chapter 7)

  As per the discussion in Section 1.2.1.3, due to various reasons such as data sizes, spatially small features, artifacts, and noise within data, the resulting feature evolution details can become nearly unmanageably large and difficult to understand. Therefore, it is essential to reduce the visual complexity of feature evolution details to fully understand the underlying trends in data sets.

  Chapter 7 presents several strategies to reduce the visual complexity within tracking graphs. Specifically, using two graph subselection approaches, filtering and feature selecting, subgraphs are extracted from a tracking graph in terms of either space or time. These approaches provide the flexibility to isolate interesting feature tracks and suppress small spurious structures in tracking graphs. This chapter further presents a novel approach that exploits the flexibility in defining temporally and spatially varying parameters for simplifying tracking graphs to promote new scientific insights. Specifically, a progressive three-pass layout algorithm that reduces the visual complexity of tracking graphs by progressive adaptation of parameter values is introduced.

- **A novel visualization and analysis environment to explore feature evolution in a generic manner** (Part IV, Chapter 8)

  Section 1.2.3 discusses various shortcomings of the existing approaches and puts forth the need for a generic approach to understand feature evolution across time. The research done as part of this dissertation has led to a new visualization and analysis environment that can understand feature evolution across time in a more general fashion. Specifically, this novel framework couples the aforementioned multiresolution unified spatiotemporal representation of features with progressive layout and visualization strategies to provide an interactive exploration of feature evolution across time, and its in-depth details are discussed in **Chapter 8**. The generalization

within this system is maintained to ensure that the framework is applicable across a variety of application domains – rather than focusing on one specific domain – and is achieved through abstraction. Given the appropriate abstractions for the unified spatiotemporal representation of features, this framework demonstrates that the challenge reverts to finding good visual metaphors independent of the data type or the community providing the solution, which then leads to more general approaches instead of more specialized ones.

1.4 Dissertation Structure

The structure of the remaining chapters in this dissertation is outlined below:

- **Chapter 2** discusses a subset of the relevant related work in the areas of feature extraction, feature correspondence, graph layouts, and dynamic data visualization.
- **Chapter 3** presents a hierarchical data representation that encodes the clustering hierarchy of univariate features for a range of parameter values.
- **Chapter 4** introduces bivariate graph, a novel compact data representation for encoding the clustering hierarchy of bivariate features for a range of parameter values.
- **Chapter 5** introduces meta graph, a novel and flexible structure that encodes feature correspondences for a range of parameter values.
- **Chapter 6** introduces a novel progressive graph layout and visualization strategy to allow interactive visualization of feature evolution across time.
- **Chapter 7** discusses subselection-based and localized, per-feature parameter value-based strategies to reduce the visual complexity of feature evolution across time.
- **Chapter 8** presents a novel visualization and analysis environment for exploring feature evolution across time in a generic manner.
- **Chapter 9** presents comprehensible details about several case studies to demonstrate the applicability and generality of the approaches and techniques presented in this dissertation.
- **Chapter 10** concludes the dissertation and discusses potential future directions.

1.5 Fundamentals and Definitions

In this section, some important definitions that provide foundations for understanding the rest of the dissertation are presented.

- **Time** is the the dimension in which the states of a system can change.
- **Time Step** is a snapshot in time.
- **Timeline** is a graphic representation of the passage of time as a line.
- **Dynamic Data** refers to data that vary over time.
- **Change** is the instance of becoming something different.
- **Continuous Change** is a change, C, that occurs at time t_c such that the limit of f of t as t approaches t_c is equal to C (i.e., $\lim_{t \to t_c} f(t) = C$).
- **Discontinuous Change** is a change, C, that occurs at time t_c such that the limit of f of t as t approaches t_c is *not* equal to C (i.e., $\lim_{t \to t_c} f(t) \neq C$).
- **Event** is the time at which a discontinuous change occurs.
- **Evolution** is a process of change along time.
- **Feature** is a distinctive attribute.
- **Nested Feature** is a feature that resides within another feature.
- **Univariate Feature** is a feature that is defined based on a single parameter value.
- **Bivariate Feature** is a feature that is defined based on two parameter values.
- **Feature Lifetime** is the parameter range within which a feature exists.
- **Feature Correspondence** is a close equivalence across features.
- **Feature Tracking** is following features that are corresponding in different time steps.
- **Feature Evolution** is a gradual change of a feature across time.

- **Feature Hierarchy** is a set of features that are grouped according to a scale parameter such that the features are represented as being above, below, or at the same level as one another.

- **Nested Feature Hierarchy** is a feature hierarchy where each feature in the hierarchy is contained in all its ancestors and contains all its descendants.

- **Disjoint Feature Hierarchy** is a feature hierarchy where each feature in the hierarchy has only one parent.

- **Disjoint Nested Feature Hierarchy** is a disjoint feature hierarchy that is nested.

- **Overlapping Feature Hierarchy** is a feature hierarchy where at least one feature in the hierarchy contains multiple parents.

- **Overlapping Nested Feature Hierarchy** is an overlapping feature hierarchy that is nested.

- **Partitioning Feature Hierarchy** is a disjoint nested feature hierarchy where the union of all features at any parameter value partitions the entire data.

- **Subsetting Feature Hierarchy** is a disjoint nested feature hierarchy where the union of all features covers only a subset of the data.

- **Graph** is a set of nodes and edges.

- **Bivariate Graph** is a graph where each node represents a bivariate feature at a specific parameter combination in a time step, and the edges represent connections to bivariate features at neighboring parameter values.

- **Tracking Graph** is a graph where nodes represent features at a specific parameter (either fixed or varying over time) within a set of time steps, and the edges represent the feature correspondences in successive time steps.

- **Meta Graph** is the union of the tracking graphs constructed for the full range of parameter values within all time steps of a data set.

1.6 Related Publications and Presentations

The research corresponding to this dissertation has been published as the following conference proceedings and presentations.

1.6.1 Peer-Reviewed Conference Proceedings

- W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci, "Interactive exploration of large-scale time-varying data using dynamic tracking graphs," in *2012 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, Oct 2012, p. 917. (*Nominated for Best Paper*).

- W. Widanagamaachchi, P.-T. Bremer, C. Sewell, L.-T. Lo, J. Ahrens, and V. Pascucci, "Data-parallel halo finding with variable linking lengths," in *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*, Nov 2014, p. 2734.

- W. Widanagamaachchi, P. Klacansky, H. Kolla, A. Bhagatwala, J. Chen, V. Pascucci, and P. T. Bremer, "Tracking features in embedded surfaces: Understanding extinction in turbulent combustion," in *2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, Oct 2015, p. 916.

- W. Widanagamaachchi, K. Hammond, L.-T. Lo, B. Wirth, F. Samsel, E. Bertini, J. Kennedy, and E. Puppo, "Visualization and analysis of large-scale atomistic simulations of plasma-surface interactions," in *Eurographics Conference on Visualization (EuroVis)-Short Papers*. The Eurographics Association, 2015.

- W. Widanagamaachchi, A. Jacques, B. Wang, E. Crosman, P.-T. Bremer, V. Pascucci, and J. Horel, "Exploring the evolution of pressure-perturbations to understand atmospheric phenomena," in *Proceedings of 2017 IEEE Pacific Visualization Symposium (PacificVis)*, Apr 2017.

1.6.2 Papers Under Review

- W. Widanagamaachchi, Y. Livnat, P.-T. Bremer, S. Duvall, and V. Pascucci, "Interactive visualization and exploration of patient progression in a hospital setting,' in *Proceedings of the AMIA 2017 Annual Symposium*, 2017.

- W. Widanagamaachchi, P.-T. Bremer, and V. Pascucci, "Combining nested feature hierarchies for bivariate feature exploration," in *Proceedings of the 2017 Conference on Visualization*, 2017.

1.6.3 Presentations

- W. N. Widanagamaachchi and V. Pascucci, "Understanding feature evolution over time for large-scale time-varying data sets," *Doctoral Colloquium*, VisWeek, 2015.

- W. N. Widanagamaachchi, Y. Livnat, P.-T. Bremer, S. Duvall, and V. Pascucci, "Interactive visualization and exploration of patient progression in a hospital setting," *Workshop on Visual Analytics in Healthcare*, 2016.

CHAPTER 2

RELATED WORK

Over the past decades, numerous techniques have been proposed in the areas of feature extraction, feature correspondence, graph layouts, and dynamic data visualization. Here, a subset of the relevant related work is discussed to provide context and background for the dissertation.

2.1 Defining and Extracting Feature Hierarchies

Feature extraction has become an apparent need for many areas of research, including machine learning, image processing, computer vision, data mining, pattern recognition, and data science. As a result, a large variety of feature definitions are found in the literature for the purpose of using feature-based approaches for the analysis of data [45–50]. Due to the diverse variety of feature-based approaches available, performing a comparative study or even a cursory glance of techniques is a very difficult task.

In any event, many of these feature-based techniques require re-processing of the entire data set for every change to the parameters associated with features, so they are too costly to apply, especially within an interactive setting. Instead, techniques that are able to extract feature information for all or a large range of parameter values in a single pass are of significant interest. Such techniques often combine a set of initial features according to a scale parameter, and thus result in hierarchical representations [51]. As the focus of this dissertation is the exploration of such feature hierarchies, this section limits the focus to the subset of the feature-based approaches involving a hierarchy of features. In this context, a number of topological techniques, clustering methods, and other types of feature-based techniques that provide threshold-dependent feature definitions have been used to effectively capture flexible feature hierarchies.

2.1.1 Univariate Feature Hierarchies

A number of popular feature extraction techniques, such as topological abstractions and clustering, naturally give rise to univariate feature hierarchies. In these hierarchies, one first extracts features at some smallest scale of interest (e.g., clusters, sublevel sets), and then iteratively combines these according to some importance metric, typically a similarity or error threshold. This section provides details on relevant related work in which such univariate feature hierarchies are explored.

2.1.1.1 Topology-Based Hierarchies

Recent advances in topological analysis have resulted in techniques that are able to efficiently extract and encode entire feature families in a single analysis pass. In most cases, concepts of Morse theory [52], [53] have been used to identify similar sets of features where the well-developed notion of simplification provided by Morse theory makes the resulting feature definitions hierarchical. Specifically, topological techniques such as Reeb graphs [34], contour trees [13], merge trees [14], and Morse-Smale complexes [34] have been used to capture hierarchical representations of features.

For a smooth simply connected manifold M and a function $f : M \to R$, the *level set* of f at the isovalue s, $L(s)$ is defined as the collection of all points in M with the function value $s : L(s) = \{p \in M | f(p) = s\}$. Its *superlevel set* is of the form $L^+(s) = \{p \in M | f(p) \geq s\}$ and *sublevel set* $L^-(s) = \{p \in M | f(p) \leq s\}$. A *contour* is defined as a connected component of a level set.

2.1.1.1.1 Reeb graphs. Reeb graphs represent the topological structure of a manifold based on a scalar-valued, sufficiently smooth function f, defined on it. Specifically, the Reeb graph of f stores information about the level sets of f and is constructed by contracting the contours of f into points, see Figure 2.1. As it describes the structure of neighboring disjoint features, the Reeb graph results in a disjoint non-nested feature hierarchy. Each node in a Reeb graph represents a contour passing through a critical point of f, and its edges the remaining contours. The Reeb graph is considered to be a very useful data structure to accelerate the extraction of level sets, and many algorithms for computing Reeb graphs can be found in the literature [54], [55].

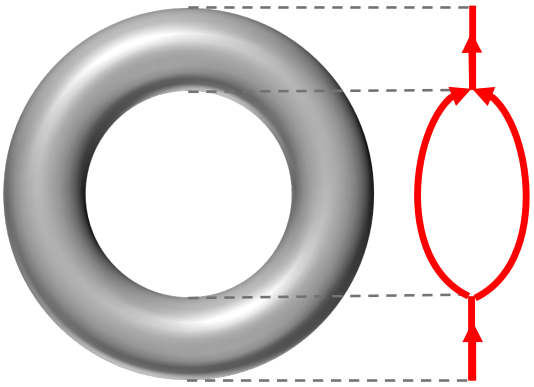
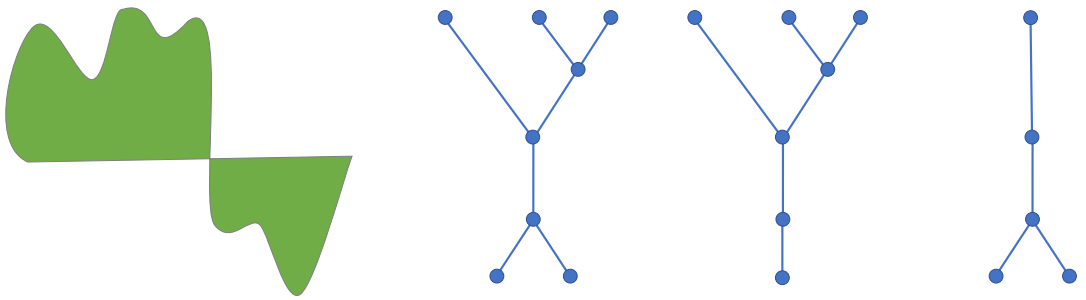


Figure 2.1: Reeb graph of the height function on a torus.

2.1.1.1.2 Contour trees. This is a graph-based representation illustrating the topological changes of level sets over the scalar value range, see Figure 2.2b. In particular, a contour tree is considered as a special case of the Reeb graph defined over a simply connected domain. As such, it also results in a disjoint non-nested feature hierarchy. To elaborate, each cut through the contour tree represents a level set, yet two level sets at different isovalues are not the union of other level sets but rather disjoint features. Initially, contour trees were used to study terrain maps [56], [57]. Today, they have been successfully used in the analysis of many other application areas, including fluid mixing, ocean science, and combustion simulations [58], [59]. Carr's thesis [60] provides a valuable resource for understanding contour tree definitions and algorithms while also demonstrating a wide range of applications of contour trees for data visualization. Again,



(a) (b) Contour tree (c) Merge tree (d) Split tree

Figure 2.2: For the surface in (a), its corresponding (b) contour tree, (c) merge tree, and (d) split tree.

just as with Reeb graphs, many approaches for computing contour trees can be found in the literature [59], [61].

2.1.1.1.3 Merge trees and split trees. The merge tree is a substructure of the contour tree that encodes the hierarchy of connected components of superlevel sets. Each node in the merge tree is a node in the contour tree, and each of its edges represents a union of components from the contour tree, see Figure 2.2c. In other words, the merging of contours as the isovalue s is swept from top-to-bottom through the full range of f is represented by the merge tree. This representation is ideally suited to encode threshold-based features and to quantize the space of features to those involving function values of critical points [62], [14]. The split tree, the other substructure of the contour tree, stores the splitting of contours through the full value range, see Figure 2.2d. Here, as they encode the superlevel sets that are nested, in contrast to the Reeb graph and contour tree, the merge tree and split tree result in disjoint nested feature hierarchies.

2.1.1.1.4 Morse-Smale complexes. Instead of looking at the behavior of level sets as in the aforementioned representations, the Morse-Smale complex describes the topology of a function by clustering the points in the domain into regions of uniform gradient flow. Each node in the Morse-Smale complex is a saddle or a minima of f, and the edges represent the integral lines connecting saddles to minimas. This structure is able to present a map of the feature space defined by the gradient flow properties and is orders of magnitude smaller than the original data. Consequently, it is considered to be very useful in many applications [63], [64], [34].

2.1.1.2 Hierarchical Clustering

Hierarchical clustering partitions data into homogeneous groups based on a measure of similarity through the use of clustering. Over the years, these techniques have received increasing attention in many different areas of research [65–69]. Consequently, a wide variety of clustering methods can be found [70]. Many sequential and parallel algorithms for hierarchical clustering can also be found in the literature. Several important results on sequential algorithms are presented in [71], [72], and details on previous parallel algorithms for hierarchical clustering are summarized in [73].

Existing hierarchical clustering techniques can be categorized into two strategies: ag-

glomerative and divisive [74]. The former strategy, agglomerative hierarchical clustering, consists of a bottom-up approach. It starts with each feature in its own cluster and merges the most similar pairs of clusters to build up a hierarchy. In the second strategy, divisive hierarchical clustering, all features are considered to be in one cluster at the beginning, and it is then split recursively to build the hierarchy in a top-down fashion.

One of the main advantages of hierarchical clustering is that any valid measurement of similarity can be used to determine the cluster similarity. Furthermore, given a valid similarity measurement between two features, many choices are available to define the intergroup similarity. The three most popular methods are single linkage (single-nearest distance), complete linkage (complete-farthest distance), and group average (average-average distance). Here, it is important to note that, even with the same cluster similarity choice, the results can lead to very different feature hierarchies depending on the group similarity choice selected. Hierarchical clustering techniques tend to impose a hierarchical structure on the underlying data irrespective of whether such a structure is appropriate. Therefore, to maintain effectiveness, it is important to make sure such a hierarchical structure is applicable to a particular data set. All in all, due to its simplicity, many applications [75], [11], [12], [76] have used this method to explore the clustering hierarchy of features.

2.1.2 Multivariate Feature Hierarchies

As we gain the ability to both produce and simulate complex phenomena with ever-increasing resolution, multivariate data sets are becoming increasingly common. Understanding such multivariate data is an integral part of science and engineering as well as many real-world phenomena. Such understanding requires investigating complex interactions and relationships associated with different variables in the data set and is usually achieved through simultaneous exploration of several variables. However, as multivariate data analysis involves the exploration of a number of variables, techniques intended for univariate features are not sufficient to effectively analyze such multivariate data sets. Therefore, new alternatives to existing univariate feature-based approaches are needed.

2.1.2.1 Topology-Based Hierarchies

Multivariate topological feature analysis is a fairly new research field. Among the few related works found on this topic, Reeb spaces [77], multidimensional Reeb graphs [78], layered Reeb graphs [79], and joint contour nets [80] can be considered as extensions of topology-based hierarchical structures for multivariate analysis.

2.1.2.1.1 Reeb spaces. The notion of Reeb space, the generalization of the concept of the Reeb graph, was first introduced by Edelsbrunner et al. [77]. Reeb space represents the topological structure of a 3-manifold M based on two scalar-valued functions, f and g, defined on it. Just as with the Reeb graph, the Reeb space of M with respect to f and g is obtained by contracting every contour to a point, maintaining adjacency between level sets. Therefore, similar to its univariate counter part, Reeb space does not describe nested features but rather the structure of neighboring disjoint features. The Reeb space construction algorithm presented by Edelsbrunner et al. [77] is complex and mathematically formalized and also lacks implementation and asymptotic analysis. Furthermore, their algorithm is confirmed to work for only four or fewer variables. Recently, Tierny and Carr presented the first practical, output-sensitive algorithm for computing the exact Reeb space for the bivariate case [81]. Conceptually, this is the equivalent of the Reeb graph for bivariate functions and encodes how the intersection of level set of two functions – called fibers – are related. Nevertheless, there is still a lack of efficient data-structure and an algorithm for computing Reeb spaces in practical applications. Other approaches such as multidimensional Reeb graphs [78] and layered Reeb graphs [79] have also been introduced for extending the Reeb graph for multivariate analysis.

2.1.2.1.2 Joint contour nets. Recent work by Carr and Duke [80] introduced a generalization for the contour tree, named joint contour nets. This generalization is also considered as an approximate representation of the Reeb space. Their work makes use of a range-based quantization approach. Specifically, contour lines are used to discretize the domain into slabs with constant scalar values, and then a graph is constructed to encode the adjacency information of these slabs. However, their construction algorithm is an approximation and not parameter free.

2.1.2.2 Hierarchical Clustering

Over the years, various multivariate clustering algorithms have been introduced that extend the existing univariate clustering techniques to multivariate data analysis [82–84]. typically based upon various multivariate distance metrics [85–88]. Although these approaches are able to cluster multivariate data, many define a joint metric that combines all parameters into a single indicator function. This joint value is often unintuitive to use, and the metrics are chosen ad hoc. More importantly, the metric's feature representation lacks the ability to independently explore each parameter space.

2.2 Feature Correspondence in Computer Vision and Visualization

Feature correspondence is one of the fundamental problems in computer vision and visualization and is a key ingredient in a wide range of applications, including object recognition, three-dimensional (3D) reconstruction, mosaicking, motion segmentation, and image morphing. Methods used for corresponding features within the computer vision and visualization literature can be broadly classified into two categories: region overlap- and attribute-based correspondences.

2.2.1 Region Overlap-Based Correspondences

The region overlap-based approach by Silver and Wang [15], [16] is considered to be the first approach of this kind. For a data set with sufficient temporal resolution, they observed that features in neighboring time steps of that data set usually overlap in space. In consequence of this observation, their approach identifies and stores the features using appropriate data structures, and then computes the correspondences using a two-stage process: an overlap test and a best matching test. In the overlap test, the corresponding features and their amount of overlap are computed. Here, oct-trees and other linked list data structures are used to accelerate the computation. Next, the best matching test compares the ratio of the region overlap amount versus the average volume among all combinations of overlapped features, and considers the combination of the maximum ratio as the corresponding features.

Later, Chen et al. [89] extended the work of Silver and Wang for tracking features in distributed adaptive mesh refinement (AMR) data sets within a distributed computing en-

vironment. Using a region overlap matching similar to that used by Silver and Wang, Sohn and Bajaj [13] also introduced a hybrid approach for defining correspondences between contour trees. Furthermore, in determining the region-based correspondences, apart from overlap matching, minimization of an affine transformation matrix has also been used [90].

2.2.2 Attribute-Based Correspondences

With attribute-based correspondences, features are represented by their attributes such as centroid, mass, and volume. These attributes are used as a simplified model to characterize the features and to identify their correspondences across time. Many approaches in the computer vision and visualization literature make use of attribute-based correspondences to explore and track features. For example, by establishing trajectories in the attribute space to determine path coherence, Sethi et al. [91] introduced a method for tracking line and region tokens. Samtaney et al. [17] applied methods from object tracking in image processing for feature tracking. In doing so, they made use of feature attributes such as centroid, mass, volume, and moment between time.

By combining these two above-mentioned studies, Reinders et al. [18] introduced an attribute-based tracking algorithm using a prediction scheme. They compute a set of attributes such as centroid, volume, mass, best fitting ellipsoid, and the skeleton of a feature. In the prediction step, a set of candidate features is computed for each feature after testing their attribute values with a number of correspondence criteria. For flexibility, each correspondence criteria is associated with a user-defined tolerance and a weight.

2.2.3 Other Types of Feature Correspondences

In the computer vision and visualization community, motion tracking has also been used to efficiently track two-dimensional (2D) objects from videos [92–96]. One initial motion tracking-based approach is the method presented by Horn and Schunck [92], [95]. Here, optical flow has been used to track 2D image features over video sequences. Since its introduction, many other techniques have been produced for calculating optical flows and tracking 2D image features.

Computing feature correspondences via graph matching is another area of research that has been explored over the years [97–100]. These approaches usually formulate the feature correspondence problem as an attributed graph-matching problem, where nodes

of the graph correspond to local features and edges to relational aspects between features, and compute the correspondences between nodes of the two graphs.

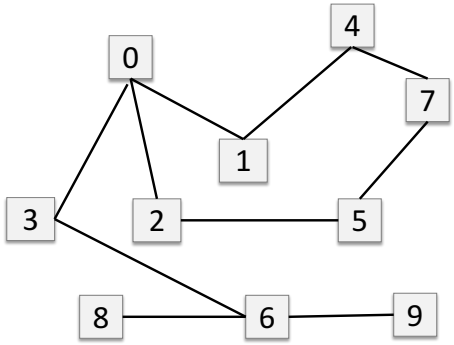
2.3 Graph Drawing for Dynamic Data

Visualizing large sets of nodes and edges is a growing need in many areas, such as biology, information retrieval, social networks, and telecommunication. As an independent field, graph drawing plays an increasingly important role in understanding such data and their underlying trends. One of the major challenges within the graph drawing community is the efficient and effective computation of graph layouts. In this context, various graph drawing methods such as planar, arc, circular, spectral, layered, orthogonal, and force directed layouts have been proposed [101]. However, these graph drawing methods are limited to static graphs.

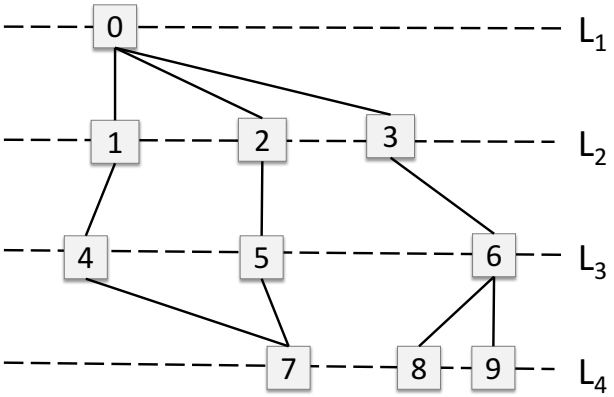
Today, many applications require the ability to handle dynamic graph layouts where the structure of the nodes and edges as well as their attributes are changing over time. Such dynamic graph layout methods require the ability to represent the evolution of relationships between nodes using readable, scalable, and effective diagrams [102–104]. Here, the use of layered graph layouts is one interesting research direction found within the dynamic graph drawing literature. As presented by Beck et al. [103], overall, methods used to visualize dynamic graphs can be broadly categorized into three types of approaches: animation-based, timeline-based, and hybrid.

2.3.1 Layered Graph Drawing

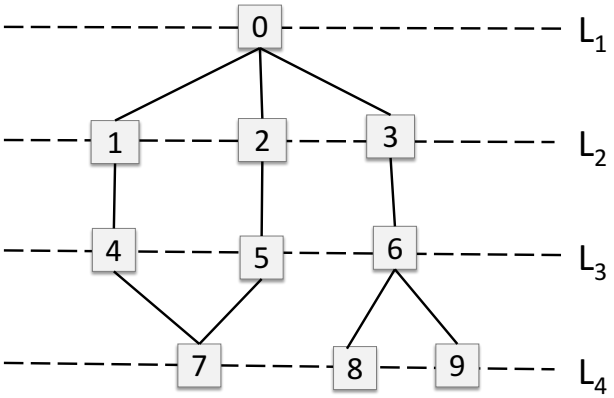
Layered graph layouts, also called 'hierarchic layouts', are mostly suited for drawing directed acyclic graphs (DAGs). First, the graph is partitioned into several layers of independent node sets, and then the interlayer edge connections and ordering of the nodes are determined according to the layout algorithm, see Figure 2.3. Many of these layered graph layout algorithms try to minimize the number of edge crossings within the layout while still maintaining the hierarchy structure. As a result, they do not guarantee a solution with an absolute minimum number of crossings, but only a local minimum [105], [106]. The main advantages of this layout are its simplicity and scalability; however, it presents a problem when arranging a large number of nodes in a limited area.



(a)



(b)



(c)

Figure 2.3: An example of a layered graph drawing. For the graph in (a), one of its possible layered graph drawings is shown in (b). Here, the graph is partitioned into several layers of independent node sets. The graph after minimizing the edge crossings with median heuristic is shown in (c).

In 1977, Warfield [107] introduced this idea of layered drawings, and since then many other approaches [108], [109] have been introduced on this topic, often with slight variations in the edge crossing minimization step. Of these approaches, the method proposed by Sugiyama et al. [109] is considered to be the most popular for computing layered graph layouts for directed graphs.

Their algorithm consists of four steps. First, by reversing the edges that cause a directed cycle, the input graph is transformed into a DAG. Then, the nodes are partitioned into horizontal layers. In the third step, the number of edge crossings is minimized by moving the nodes horizontally, thus fixing their horizontal placement. Finally, the previously reversed edges are inserted back into their original orientation, and the final graph layout is produced. In addition to these steps, this algorithm contains a separate restriction that makes sure long edges in the graph (i.e., edges across nodes separated by several layers) are as straight as possible within the layout. As a testimony to its popularity, many modifications and improvements to this algorithm can be found in the literature. Furthermore, systems such as da Vinci [110], Dot [35], and GraphLet [111] have included this method within their framework for drawing directed graphs.

Minimizing the number of edge crossings within a layered graph is NP-hard even for only two layers. For this problem of two-layer crossing minimization, various heuristics such as Barycenter [109], Median [112], Greedy-insert [113], Greedy-switch [113], and Sifting [114] have been proposed in the literature. As the name suggests, the barycenter heuristic reorders the nodes according to the average values of their neighbor's order, and the median heuristic reorders according to the median value, see Figure 2.3c. The greedy-insert and greedy-switch heuristics either insert or switch nodes based on a greedy approach to minimize the number of edge crossings in the layout. Although these heuristics are simple to implement, suited for obtaining progressive results, and offer speedy execution at run-time, they are inefficient in several other aspects. They often use a layer-by-layer sweep to reduce crossings in the graph. That is, they push crossings downward or upward in the graph until they are resolved at layer k, so the edge crossings get swept across layers. Also, as they are restricted to a local view, they can get stuck in a local minimum, leading to suboptimal solutions.

2.3.2 Animation-Based Graph Drawing

As the name suggests, these types of graph drawings use animation throughout the visualization. Specifically, using a time-to-time mapping [103], the time steps of the dynamic data set are mapped to a sequence of graphs in an animation. This mapping of the time steps assigned to the sequence of graphs is used to visualize the graph results in an animated representation. Additionally, in order to effectively convey the feature evolution details, most animation-based approaches maintain a mental map. The graph layout algorithm preserves this mental map while drawing the sequence of graphs.

Due to its simplicity and intuitiveness, many approaches make use of animation-based graph drawings for dynamic data sets [115–119]. However, as animation-based graph drawings look at the time steps of the data set one at a time, they can lead to high cognitive load, especially for larger data sets. As a result, it can be difficult to follow a particular feature across long periods of time, making it nearly impossible to identify underlying trends in data.

2.3.3 Timeline-Based Graph Drawing

These types of graph drawings represent dynamic data using a timeline. In particular, for a given dynamic data set, a graph is drawn onto a timeline using a time-to-space mapping [103]. Here, instead of viewing one graph at a time, as in animation-based approaches, the complete sequence of graphs is displayed in one visualization. Therefore, these approaches have the potential to present better overviews of dynamic data sets. However, as only a small amount of space is available for drawing each graph in the sequence, these approaches involve a visual scalability problem. In other words, as the data sizes increase, maintaining the comprehensibility of the visualization becomes increasingly difficult. Again, many approaches that make use of timeline-based approaches can be found in the literature [120–124]. Tracking graphs also falls under this category [18], [125].

2.3.4 Hybrid Graph Drawing

Although most existing dynamic graph drawing techniques can be unambiguously classified as animation- or timeline-based, a few approaches that combine both techniques can be found in the literature [126], [127], [32]. Usually, these hybrid approaches combine both time mappings, time-to-time and time-to-space, in a more closely connected manner.

The work by Sallaberry et al. [127] presents a timeline-based, aggregated representation to illustrate cluster evolution across time while also providing animated navigation capabilities to explore each time step. Similarly, Hadlak et al. [126] embed animated graph layouts into a timeline. In parallel edge splatting [32], long sequences of graphs are animated as a moving timeline representation.

2.4 Dynamic Data Visualization

The increasing amount of data sets available today creates new challenges for developing effective visualization techniques, especially for dynamic data sets. Due to their dynamic nature, general data visualizations differ considerably from dynamic data visualizations. The most common means of analyzing dynamic data is through exploration of feature evolution within data. Traditionally, snapshots of individual time steps or animations have been used for visualizing the evolution of features. As the field of visualization has grown, depending on the subject area, very different techniques have been developed for understanding the evolution of features within dynamic data, often with a strong separation into SciVis and InfoVis categories. This section starts with the relevant related work in SciVis and InfoVis before describing some 'scientific-information' systems that focus on combining both viewpoints.

2.4.1 Scientific Visualization Systems

In contrast to related InfoVis research, the SciVis focus is often on managing data efficiently to produce scalable solutions [26], [128], [14], [129] rather than on the clarity or effectiveness of the visualization. This data efficiency is usually achieved through optimized data structures [130], compression techniques [24], or advanced hardware [25].

Past techniques such as illustration, abstraction, art, morphing, and animation have been used for effective and expressive visualization of feature evolution [20–22], [131]. Unfortunately, these techniques often do not scale to larger data sets and tend to be highly specialized to particular use cases. For example, [21] presents a set of illustrative visualizations for 3D flows by consolidating multiple time steps into one visualization. Although this approach produces interesting visualizations, similar to hand-crafted illustrations, those visualizations quickly become cluttered as the numbers of features increase. Likewise,

morphing-based tracking by Muelder and Ma [20] allows interactive feature selection, visualization, and refinement at low computational costs, but it is useful only for tracking individual features and cannot explicitly handle different feature events such as births, splits, and merges.

In addition to these techniques, many other diverse techniques have also been introduced for visualizing feature evolution across time [132–135]. Particularly, tracking graphs [18], [125], with which feature evolution is displayed as a collection of feature tracks, are considered to be one of the effective visualization methods for feature evolution exploration. Using a timeline-based approach, these graphs provide global overviews of the entire dynamic data set in an efficient manner. Nevertheless, tracking graphs can also become complex for larger data sets, but they lend themselves more easily to filtering, and thus produce high-level overviews.

2.4.2 Information Visualization Systems

InfoVis has a similarly long history of research depicting feature evolution but is aimed at very different data sets, such as social media, financial, video, web, financial, healthcare, and source code data. Identifying and tracking events [27], [136], [137], exploring topic trends [28], and analyzing topic competitions [30] data are several research directions found in InfoVis.

One of the early examples of these studies is the ThemeRiver [42]; its more recent descendant is TextFlow [27]. ThemeRiver and TextFlow present thematic changes for textual data in the context of a timeline allowing one to discern patterns in individual/multiple themes relative to time. However, these studies are not scalable in terms of both data mining and visualization. Moreover, meaningful topic themes are not easily distinguishable from the visualization.

Besides textual data, a variety of dynamic data sets appear in InfoVis applications. For instance, StoryFlow [29] presents a novel way to illustrate dynamic relationships between entities in stories and movies. It presents aesthetically appealing visualizations and handles hierarchical relationships between entities over time. WireVis [138] implements a multiview approach to visualize categorical, dynamic data from financial transactions and depicts relationships among keywords and accounts. Moreover, visual analysis and sim-

plification of temporal event sequences in healthcare data are presented in EventFlow [43]. Unlike others, EventFlow has been extended to other applications such as cybersecurity, sports analytics, and incident management, yet it remains firmly routed in the InfoVis data domain. A similar example is OutFlow [44], which visualizes event progression pathways in electronic medical records and sports events. TrailExplorer2 [139] visualizes web sessions' data to reveal temporal patterns and enables data exploration at different levels of detail.

2.4.3 Systems Integrating Scientific and Information Visualization

For many years the visualization community has discussed ways of bridging the gap between the SciVis and InfoVis domains [140–144]. In fact, many approaches exist where various InfoVis techniques are applied to visualize scientific data and vice versa [145–148]. Additionally, several examples of systems that attempt to integrate scientific and information visualization can also be found in the literature [149–154].

In particular, the Titan project [153], which systematically unifies InfoVis and SciVis, and the model-based visualization taxonomy introduced by Tory and Möller [149], are of particular interest due to the originality of the approach. Titan [153] includes various informatics-oriented functionalities, database access, graph algorithms, graph layouts, and charts and even makes use of parallel processing and client/server capabilities of VTK for data distribution. Since Titan is integrated into VTK, it inherits many qualities of that toolkit as well. However, most of its informatics-oriented functionality is conducted in a serial fashion, and it has limited capabilities for exploring evolution of features. The work by Tory and Möller [149] is based on the data model rather than the data themselves. Specifically, in order to distinguish between SciVis and InfoVis data, their visualization algorithms are classified by whether the data model is discrete or continuous.

PART II

INTERACTIVE EXTRACTION OF FEATURE EVOLUTION

CHAPTER 3

COMPUTING AND ENCODING UNIVARIATE FEATURES AT MULTIPLE SCALES

The first step toward understanding a dynamic data set via exploring the evolution of features is defining, identifying, and extracting its features. Given a dynamic data set, this exploration requires the feature of interest to be defined and then all features from all time steps extracted. For most data sets, the optimal parameter value for extracting features is not known in advance, and therefore, exploring the entire parameter range is of significant value. Such exploration has the potential to explore how a certain feature's behavior and attributes change as the parameter value varies, and thus better characterize that feature.

Nevertheless, feature computation is an expensive operation that typically requires the entire domain, or parts of it, to be inspected for each time step in the data set. Furthermore, traditional approaches often require features to be recomputed at each change in the parameter settings. Therefore, given the expected data sizes, on-the-fly feature computation is practically infeasible and requires massively parallel computing resources. The alternative is to precompute features for a wide range of potential parameters and store the results in an efficient look-up structure. This approach reduces the problem of feature computation per time step to a single preprocessing step that can be easily performed in parallel. Moreover, such an approach has the potential to enable interactive exploration of features.

Many feature definitions exist in the literature for the purpose of using feature-based approaches for the analysis of data. Among these definitions, the simplest and most commonly used are *univariate features*, which are defined using a single parameter (e.g., burning cells in combustion data are defined using the fuel consumption rate, halos in cosmology data are defined using a distance value, patient groups in healthcare data are defined using a patient similarity value, and Twitter topics in Twitter data are defined

using a textual similarity value). This chapter focuses on the problem of interactive feature exploration for univariate features. Specifically, a hierarchical feature representation that is capable of storing all possible univariate features for their entire parameter range is discussed. Here, appropriate abstractions are maintained within the feature representation in order to maintain its applicability across a range of application domains. This chapter provides comprehensive details about this feature representation and presents several application results to demonstrate its utility.

3.1 Feature Hierarchies

Precomputing all possible features and encoding them in an efficient feature representation ensure interactive exploration of features. Particularly, for a specific time step in a dynamic data set, extracting univariate features for all possible parameter values and then grouping them results in such a feature representation. By maintaining a notion of scale, this feature grouping naturally approximates a meaningful *feature hierarchy* (i.e., a set of features combined according to a scale parameter). Here, by convention, the top of the hierarchy is considered to be the direction toward the leaves and the bottom the direction toward the root of the feature hierarchy.

The naive approach to create this feature hierarchy is by exhaustively precomputing all possible univariate features at all possible scales. Nevertheless, many popular grouping algorithms also produce univariate features for varying scales, which can in turn create this hierarchy. In theory, a feature hierarchy created in this manner can be either disjoint or overlapping. A *disjoint feature hierarchy* is a hierarchy in which every feature has only one parent, whereas in an *overlapping feature hierarchy* at least one feature has multiple parents. Both of these hierarchies, whether disjoint or overlapping, can be either in nested (i.e., one object resides within another object) or non-nested form. In practice, nested disjoint feature hierarchies are most commonly found. For example, clustering techniques progressively merge elements [11], [12], and threshold-based segmentation creates increasingly larger regions [13], [14], generating nested disjoint feature hierarchies. It is important to note that disjoint hierarchies can be represented as trees, whereas overlapping hierarchies result in a graph. This proposed hierarchical feature representation is able to maintain both hierarchy forms, but certain aspects of the feature representation change depending on whether

the hierarchy is disjoint or overlapping. Therefore, for clarity, disjoint and overlapping hierarchy details within this section are discussed separately.

Furthermore, in order to utilize this type of a hierarchical feature representation across a wide range of application domains, it is important to nurture a generic representation. In this dissertation, the generic nature of the feature hierarchy is achieved through abstraction. Specifically, depending on the underlying data type of a dynamic data set, domain scientists are allowed to define a feature of interest (e.g., burning cells in combustion data and halos in cosmology data) and specify a meaningful method of grouping (e.g., threshold-based segmentation in combustion data and connectivity-based clustering in cosmology data). Maintaining abstraction for these two aspects within the feature hierarchy ensures its applicability across a wide range of application domains. Here, it is important to note that this feature of interest is univariate (e.g., burning cells based on the fuel consumption rate, halos based on a distance value).

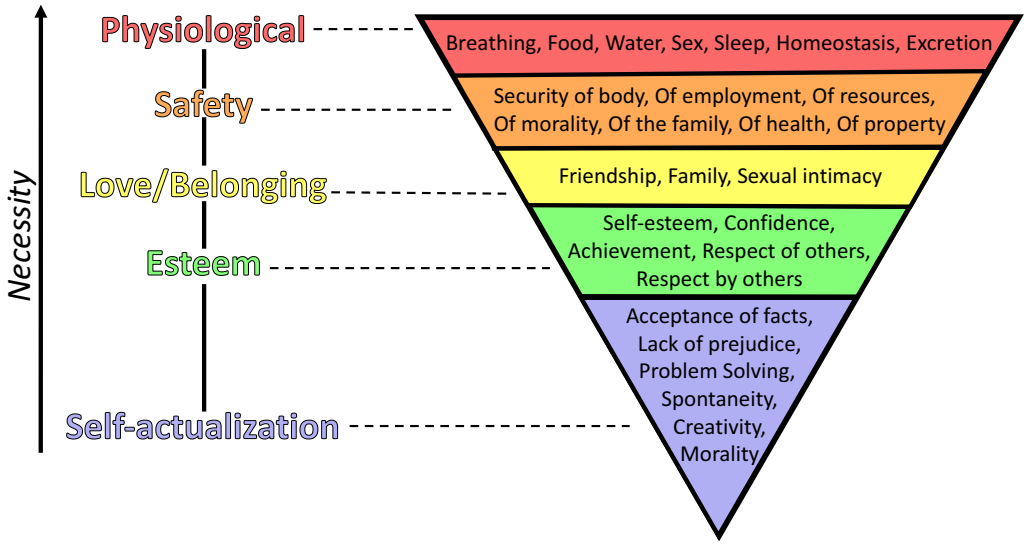
3.1.1 Hierarchy Construction

As per the above discussion, in order to create this type of a hierarchical feature representation, first, the abstract aspects should be defined. Specifically, depending on the underlying data type of a dynamic data set, the feature of interest and feature grouping method can be specified. Once these two aspects are defined, univariate features are extracted for each time step of the data set and then grouped to form the feature hierarchy. In cases where visualizing this feature hierarchy is necessary, a hierarchical graph layout (in horizontal or vertical form) can be computed for both of its forms (i.e., tree or graph) by following the depth-first ordering of its features.

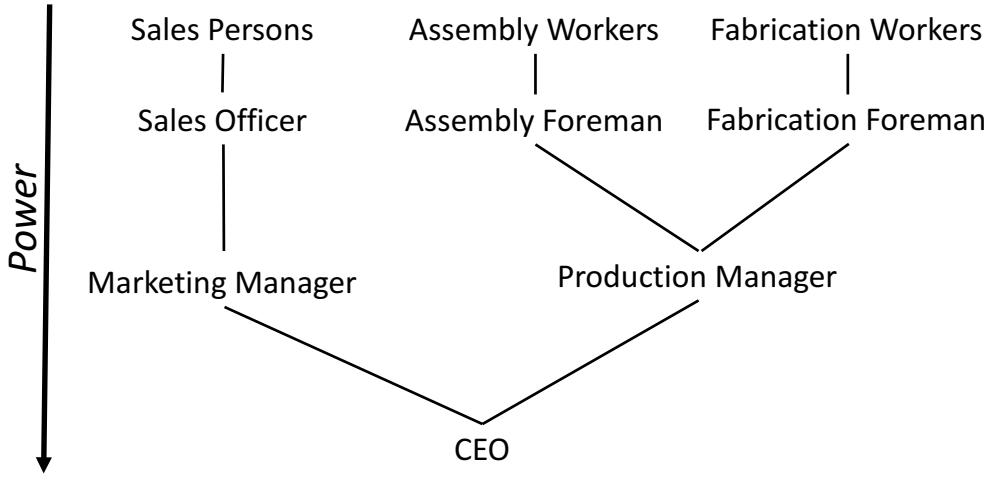
3.1.1.1 Disjoint Feature Hierarchy

Figure 3.1a presents an example of a disjoint feature hierarchy. It shows Maslow's hierarchy of needs [155], which groups different types of needs based on their fundamental level of necessity to result in a simple non-nested feature hierarchy. Figure 3.1b shows another example in which the structure of an organizational is represented using a disjoint feature hierarchy.

Each *branch* within the disjoint feature hierarchy represents a feature (i.e., a set of elements), and the parameter range of that branch defines that feature's *lifetime* within



(a) Maslow's hierarchy of needs



(b) Hierarchical organizational structure

Figure 3.1: Two examples of disjoint feature hierarchies. (a) Maslow's hierarchy of needs [155] showing how different types of needs are grouped based on their fundamental level of necessity. (b) A hierarchical structure showing how different people are grouped together based on their level of power within an organization.

the data set. In other words, a feature is considered born at the parameter value of the top node, and it is *active* until it merges with its parent past the parameter value of its bottom node. However, a merged feature is still considered *alive* even after merging. Furthermore, as indicated by the colors in Figure 3.1a, there exists a natural correspondence of branches in the disjoint feature hierarchy to its original elements (e.g., words, points, mesh vertices). Storing the segmentation information in each branch of the feature hierarchy allows one to

easily extract the geometry of a feature. Additionally, when the hierarchy is constructed, feature-based attributes such as first-order statistical moments and shape characteristics can also be computed and stored on a per-branch basis. In this fashion, this proposed feature hierarchy can encode an entire disjoint feature family alongside its geometry and feature-based attributes in a compact and efficient manner.

The nested disjoint feature hierarchy is a special case of the disjoint hierarchy. As previously mentioned, nested disjoint feature hierarchies are most commonly found in practice. Figure 3.2 shows an example of a nested disjoint feature hierarchy constructed using a connectivity-based clustering algorithm. Here, the clustering algorithm considers distance, and therefore, individual points within the domain are progressively merged, with the closest ones clustered first. Another example of a nested disjoint feature hierarchy constructed using a threshold-based segmentation is shown in Figure 3.3. Here, the merging of contours as the parameter value is swept from top to bottom through the full range is stored.

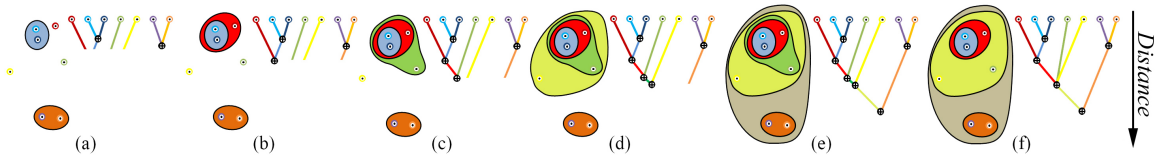


Figure 3.2: Construction of a nested disjoint feature hierarchy using a distance-based clustering algorithm: (a)-(e) A set of points is merged as the parameter value (i.e., distance) is swept top to bottom through the full value range. (f) The augmented disjoint feature hierarchy obtained by removing branches that are shorter than a certain desired interval.

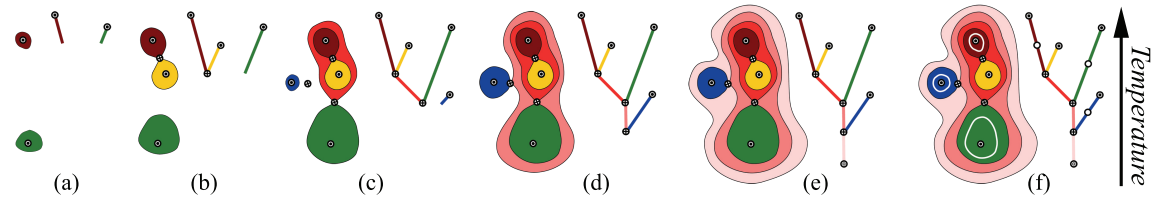


Figure 3.3: Another example of a nested disjoint feature hierarchy created with the use of threshold-based segmentation: (a)-(e) The merging of contours is recorded as the parameter value (e.g., temperature) is swept top to bottom through the full value range. (f) The augmented disjoint feature hierarchy obtained by introducing additional valence two nodes to split the branches that are longer than the desired interval.

Given a nested disjoint feature hierarchy, instead of maintaining the full segmentation of a feature on a per-branch basis, for efficiency, only its exclusive elements (i.e., those not part of its descendants) can be stored. In other words, an element can be stored only with the first feature that contains it, which reduces the data duplication within the hierarchy. However, instead of extracting the geometry of a feature from its corresponding branch as in the non-nested case, now its geometry needs to be constructed as a union of branch segmentation within that feature's entire subtree. As the nested hierarchy is constructed, the corresponding exclusive elements are stored, starting from the leaf nodes. Simultaneously, the feature-based attributes are also computed based on a feature's exclusive elements, and then accumulated along the hierarchy.

In general, disjoint feature hierarchies, whether nested or not, quantize the space of features depending on the parameter values involved. This implicit quantization is often too coarse or too fine to be practical. In such cases, an augmented disjoint feature hierarchy can be constructed. When the quantization is too fine, this augmented disjoint feature hierarchy can be created by removing branches that are shorter than a certain desired interval, see Figure 3.2f, and when the quantization is too coarse, it can be created by introducing additional valence-two nodes to split the branches that are longer than the desired interval, see Figure 3.3f.

3.1.1.2 Overlapping Feature Hierarchy

An example of an overlapping feature hierarchy that captures the level of generalization across a set of theories is shown in Figure 3.4. Due to the overlapping property, a feature within this hierarchy can have multiple parents (e.g., in Figure 3.4 node 'Linearorder' has 'Trees' and 'Lattices' as parents). Therefore, in contrast to the disjoint hierarchy, each feature within the overlapping hierarchy cannot always be represented by a single branch. As a result, instead of associating a branch with a feature as in disjoint hierarchies, in overlapping hierarchies a feature is associated with a node within the hierarchy. Specifically, for each feature within the overlapping hierarchy, its complete segmentation and other feature-based attributes are stored on a per-node basis.

For a nested overlapping hierarchy, just as in the nested disjoint case, storing only exclusive elements within a feature removes data duplication. However, due to the over-

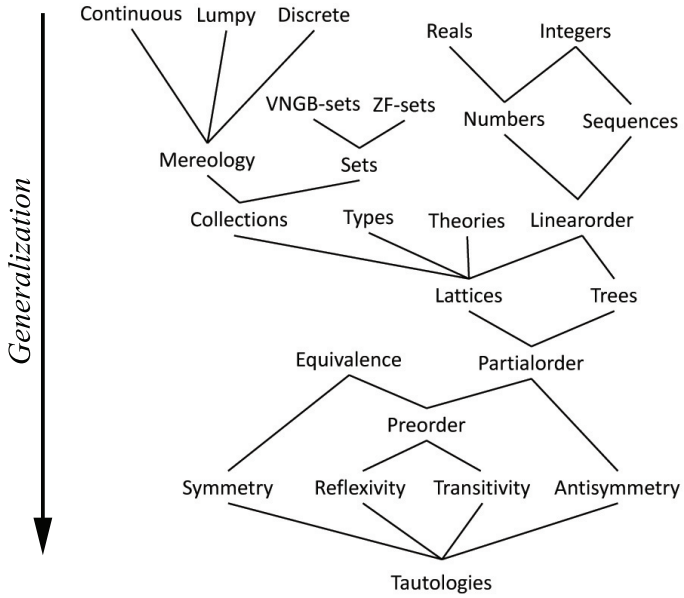


Figure 3.4: An example of an overlapping feature hierarchy. Here, a set of theories is grouped together based on their level of generalization. Each theory within the hierarchy is a specialization of the ones below it and a generalization of the ones above it.

lapping property, computing the feature-based attributes based on the exclusive elements and naively accumulating them along the hierarchy results in incorrect attribute values. Consider the same example in Figure 3.4. If this hierarchy is nested, then attributes at the 'Partialorder' node are computed by accumulating attributes of its children (i.e., 'Trees' and 'Lattices' nodes). Then, the attributes at the 'Linearorder' node would be considered twice, and thus the accumulated value at 'Partialorder' node would be incorrect. Therefore, when the hierarchy is created, instead of accumulating attributes along the hierarchy, attributes are computed by looking at a feature's entire segmentation, just as for the non-nested overlapping hierarchy.

Additionally, as overlapping feature hierarchies can also quantize the space of features depending on the parameter values involved, augmented overlapping feature hierarchies can be utilized as necessary.

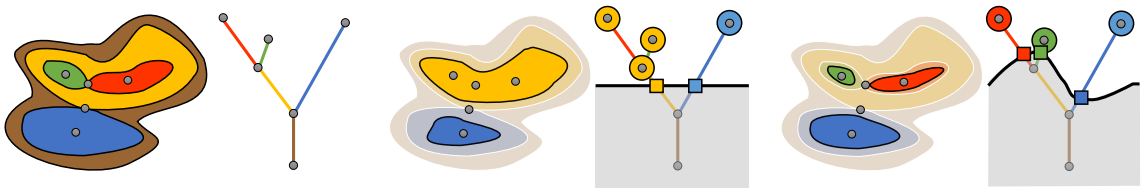
3.1.2 Feature Extraction

Once a feature hierarchy is computed, its features can be quickly and easily extracted for any parameter value within the full parameter range. Given a parameter value f_1 within the full range of f, the corresponding features can be found by cutting the hierarchy

of f at f_1. Here, it is important to note that this cut should not necessarily be a horizontal line (i.e., a fixed parameter value). The feature hierarchy permits arbitrary cuts to be made for representing localized, per-feature parameter values. Consequently, feature hierarchies facilitate a more flexible feature extraction.

3.1.2.1 Disjoint Feature Hierarchy

For the disjoint hierarchy, Figure 3.5b shows an example of features extracted by cutting the disjoint feature hierarchy of f at a fixed parameter value. This cut creates a forest, where each of its connected components (i.e., a subtree) is a feature existing at that parameter value. Furthermore, due to the hierarchy's tree structure, each subtree can be represented by its bottom, active branch. In other words, each feature existing at the selected parameter value can be represented by the active branch at the cut, and therefore, those features can be quickly extracted by looking at the branches. Furthermore, as relevant feature details are stored within the hierarchy on a per-branch-basis, in addition to the features, their attributes and geometry information can also be extracted from the hierarchy without any additional computation. For the nested case, as only its exclusive features are stored within the hierarchy, its geometry needs to be constructed as a union of branch segmentation for each feature existing at the selected parameter value. Also, as shown in Figure 3.5c, for a disjoint feature hierarchy, whether nested or not, any arbitrary cut that intersects each path from a leaf to the root at most once results in a valid segmentation.



(a) (b) (c)

Figure 3.5: Extracting features from a disjoint feature hierarchy. Features for a fixed parameter value can be extracted by cutting the hierarchy of f at that value and ignoring all pieces below, shown (a) before and (b) after extraction. Each connected component in the forest (i.e., a subtree) represents a feature existing at the selected parameter value. Similarly, using an arbitrary cut that defines per-feature, localized parameter values, more flexible features can also be extracted, as shown in (c).

3.1.2.2 Overlapping Feature Hierarchy

An example of an overlapping feature hierarchy cut at a fixed parameter value is shown in Figure 3.6. This cut results in a subgraph with multiple roots, whereas the disjoint hierarchy results in a forest. Every connected component in the subgraph, which is also a subgraph, is a feature existing at the selected parameter value. For disjoint hierarchies, the connected component details are directly encoded within the subtrees. In overlapping hierarchies, even though each connected component in the subgraph represents a feature existing at the cut, that subgraph no longer directly encodes the connected component details. Therefore, at a particular cut, a graph traversal within the resulting subgraph (toward its leafs) is needed to identify its *active* features (i.e., those features existing at the cut), as shown in Figure 3.6. In particular, for a cut, its resulting subgraph is traversed 'upwards' to extract all *alive* features. During the graph traversal, the connected components within the resulting subgraph are computed, each representing a feature existing at the cut. Along with the features, their attributes and geometry details can also be computed on a per-connected component basis. In the non-nested case, this information is computed for each connected component by combining the details of its active features. As only its

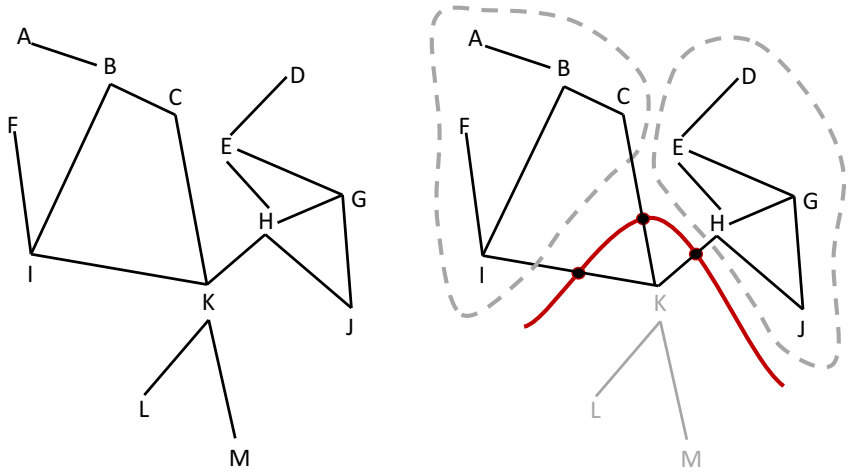


Figure 3.6: Extracting features from an overlapping feature hierarchy. Just as in the disjoint hierarchy, features for a fixed parameter value are extracted by cutting the hierarchy of f at that parameter value and ignoring all pieces below, shown (a) before and (b) after extraction. Here, each connected component (indicated in a dashed line) in the resulting subgraph is considered as a single feature.

exclusive elements are stored within a feature, a connected component's attributes and geometry details should be computed for the nested case by combining details of all its alive features.

3.1.3 Implementation Details

Given a dynamic data set, an offline preprocessing step is used to compute feature hierarchies. Specifically, a feature hierarchy is created for each time step of the data set, and then those results are stored in multiple files (i.e., one file per time step). For each feature in the hierarchy, pointers to its parents and descendants, corresponding parameter values (for all its parent branches), elements (exclusive or otherwise), and feature-based attributes are stored.

3.1.3.1 Disjoint Feature Hierarchy

When exploring features within a disjoint feature hierarchy, first, its corresponding hierarchy file is read. Then, this hierarchy information is stored within an interval tree data structure [156]. The interval tree is a tree-based data structure that can hold intervals. The interval tree can quickly extract all intervals that overlap with a given interval or point. In order to store the disjoint hierarchy information within the interval tree, all feature lifetimes (i.e., parameter ranges of branches) within the hierarchy are considered, and then those intervals are stored within the interval tree data structure. An additional look-up structure is also maintained to store the hierarchical structure of the disjoint feature hierarchy. For each feature in the hierarchy, its parameter values, parent details, and segmentation details are stored within this look-up structure. Together, these two data structures allow interactive exploration of features for their entire parameter range.

When a certain query is made, the interval tree is traversed to return all active features (i.e., all features within the query's parameter range). Each active feature represents a connected component (i.e., a feature) existing at the query parameter. As the hierarchy file stores the geometry details and feature-based attributes of each feature within the hierarchy, when extracting the active features for a particular query, these details can be extracted without any additional computation. However, for a nested disjoint hierarchy, as only the exclusive elements of a feature are stored, the entire subtree of an active feature needs to be traversed in order to obtain the complete segmentation details.

3.1.3.2 Overlapping Feature Hierarchy

In overlapping hierarchies, a feature can have multiple parents, and thus determining its lifetime is not straightforward. Therefore, instead of looking at the lifetime of a feature as in disjoint hierarchies, the birth time of each feature is considered. To explore features within an overlapping hierarchy, its details are read from the corresponding hierarchy file, and then stored in a range tree data structure [156]. A range tree is an ordered tree data structure that stores a set of points and allows quick retrieval of points within a given range. All features are stored according to their birth time in a 2D range tree to quickly extract all features that are alive at a given parameter. Just as in the disjoint case, an additional look-up structure is maintained to store the hierarchical structure of the overlapping hierarchy.

For a given parameter f, the query $(-\infty, f]$ is considered, and the range query is processed to obtain all features that are alive. In a subsequent step, these set nodes that are alive are traversed using the graph to extract their connected components. Here, each connected component represents a feature existing at parameter f. During this graph traversal, the feature-based attributes and segmentation details of each feature are computed on a per-connected component basis, either by combining the connected component's features at the cut (in the non-nested case) or combining all features (in the nested case).

3.1.4 Advantages and Limitations

These proposed feature hierarchies are significantly smaller than the original data sets, and yet retain enough information to perform the desired feature-based analysis. Therefore, with the ever-increasing data sizes, utilizing this feature representation has distinct advantages over other approaches. One main advantage is its generic nature. It can be used across a wide range of application domains. The only requirement is that the appropriate details need to be specified for the abstract aspects (i.e., the feature of interest and the feature grouping method) within the feature structure. Once these details are specified, feature hierarchies are able to encode the entire feature family for all time steps of the data set.

As the parameter value is varied, these feature hierarchies have the capability to extract

features quickly and efficiently without any feature recomputation. Once the hierarchy is computed, only a traversal within the hierarchy is needed, which takes very little time relative to the construction time, to obtain feature information for multiple parameter values, whereas with the other approaches a full re-run of the algorithm is needed for every parameter change. Therefore, when interactive exploration of the parameter range is needed, it becomes more efficient to use a feature hierarchy. Also, at the feature extraction time, both fixed and localized, per-feature parameter values can be specified, allowing for a more flexible feature definition. Furthermore, as all the necessary details are stored within the hierarchy, in addition to the features, their geometry and feature-based attributes can also be extracted or computed quickly at run time.

One limitation of this feature representation is that it is applicable for only univariate features. As only one parameter value is considered when creating the feature hierarchy, it is not sufficient to analyze multivariate features. Another limitation is that the representation tends to quantize the space of features depending on the parameter values involved. As a result, the implicit quantization can often be too coarse or too fine to be practical. However, creating an augmented feature hierarchy avoids this issue to a certain degree. Additionally, the complexity of the hierarchy construction step depends on the feature of interest and the feature grouping methods involved, and thus varies according to the underlying data type. From the research conducted as part of this dissertation, it was found that in most cases creating this hierarchical feature representation is simple and relatively straightforward. Another limitation is that due to their hierarchical nature, feature hierarchies impose a hierarchical structure on the underlying data irrespective of whether such a structure is appropriate. Therefore, it is important to ensure such a hierarchical structure is appropriate for a particular data set before any data analysis is performed using this feature representation.

3.2 Application Results

In this section, several real-word data sets are presented to demonstrate the effectiveness of the feature hierarchy. Specifically, results from cosmology, combustion, and image domains are presented.

3.2.1 Cosmology Data

Here, a cosmology data set containing 16.7 million particles from a 256^3 cosmological simulation is considered. This data set contains 241 time steps totaling about 119GB of raw data. As the data sizes increase, analyzing these cosmology data sets is becoming increasingly challenging and requires substantial computational resources. In particular, some of the baseline analyses, such as halo finding, require new scalable and flexible alternatives to existing techniques.

A 'halo' [157], [158] is defined as an over-dense region of dark matter particles and represents one of the common features of interest within cosmology data. The most common definition of a halo is based on a friends-of-friends (FOF) clustering [159]. It combines all particles that are reachable through links shorter than a predefined distance (i.e., the linking length) to be in one halo. However, although there exists a default linking length, it is well known that the halo structure can change substantially for different linking lengths, and as the structure formation in FOF halo finding is hierarchical, it results in 'halos-within-halos' (i.e., subhalos). Consequently, analyzing halos for different parameters is of significant interest [160], [161].

To that end, constructing a feature hierarchy that computes the entire halo family by hierarchically encoding halos at different linking lengths is of much value to the collaborating scientists. Such a hierarchy enables analyzing a range of linking lengths interactively, significantly increasing the flexibility overall. Here, the data-parallel FOF halo-finding algorithm introduced by Widanagamaachchi et al. [162] that is implemented using PISTON [163], a cross-platform library of data parallel visualization and analysis operators, is used to create this feature hierarchy. This halo-finding algorithm results in nested disjoint feature hierarchies. Along with the hierarchy, feature-based attributes such as halo id, size, position, and velocity information are computed and stored for each halo within the hierarchy.

According to the collaborating scientists, they are rarely interested in exploring the entire linking length range but more so the linking values around a default linking length value. Therefore, instead of creating the full clustering hierarchy of the input particles, computing the halo hierarchy for a certain linking length range is more appropriate. The aforementioned halo hierarchy creation algorithm by Widanagamaachchi et al. [162] al-

lows such flexible hierarchies to be created. For this particular data set, the default linking length value is 0.2, and thus a linking length range around this value is selected at the hierarchy construction time.

Furthermore, for this data set, the quantization of the linking length within the hierarchy was found to be too fine. For instance, even for a small subset of this data set (about 24000 particles), when considering a linking length range of 0.19 to 0.2, some particles in the final feature hierarchy contained ≈15000 parent nodes whereas others contained fewer than 10 parent nodes. Therefore, an augmented hierarchy was created by removing branches that are shorter than a certain desired interval (e.g., 0.001). Such an augmented hierarchy reduced the granularity of the clustering found within the final hierarchy and made interactive halo extraction possible.

Due to the large data sizes within the original data set (around 119GB), the preprocessing steps to compute the relevant augmented feature hierarchies were run on a single node of the Maverick system at the Texas Advanced Computing Center (TACC). Here, when creating the augmented feature hierarchies, branches shorter than 0.001 were removed. Once the augmented feature hierarchies are computed for a linking length range of 0.1 - 0.2, the data sizes are reduced to 470MB. These data sizes include the hierarchy details, feature-based attributes, and geometry information. It is important to note that most of that data size is used for storing the necessary segmentation information.

Figure 3.7 compares timing results for this feature hierarchy against the Paraview's halo-finding implementation [164] run serially and with multiple MPI processes. Here, the timing results for the feature hierarchy include both the hierarchy construction time and halo-finding time. Specifically, for comparison purposes, instead of a preprocessing step, the hierarchy is constructed during the first query. As multiple queries are performed, only a traversal within the already constructed hierarchy is needed to find halos. For a comprehensive comparison, the timing results for several feature hierarchies with different linking length ranges are presented here. As illustrated in the Figure 3.7, the performance and scaling of the hierarchy algorithm constructed for a single linking length (i.e., linking length range 0.2 - 0.2) are better than the Paraview implementation run serially and comparable when run with multiple MPI processes. As the linking length range is increased, the total hierarchy construction time increases since it requires more work, as indicated by

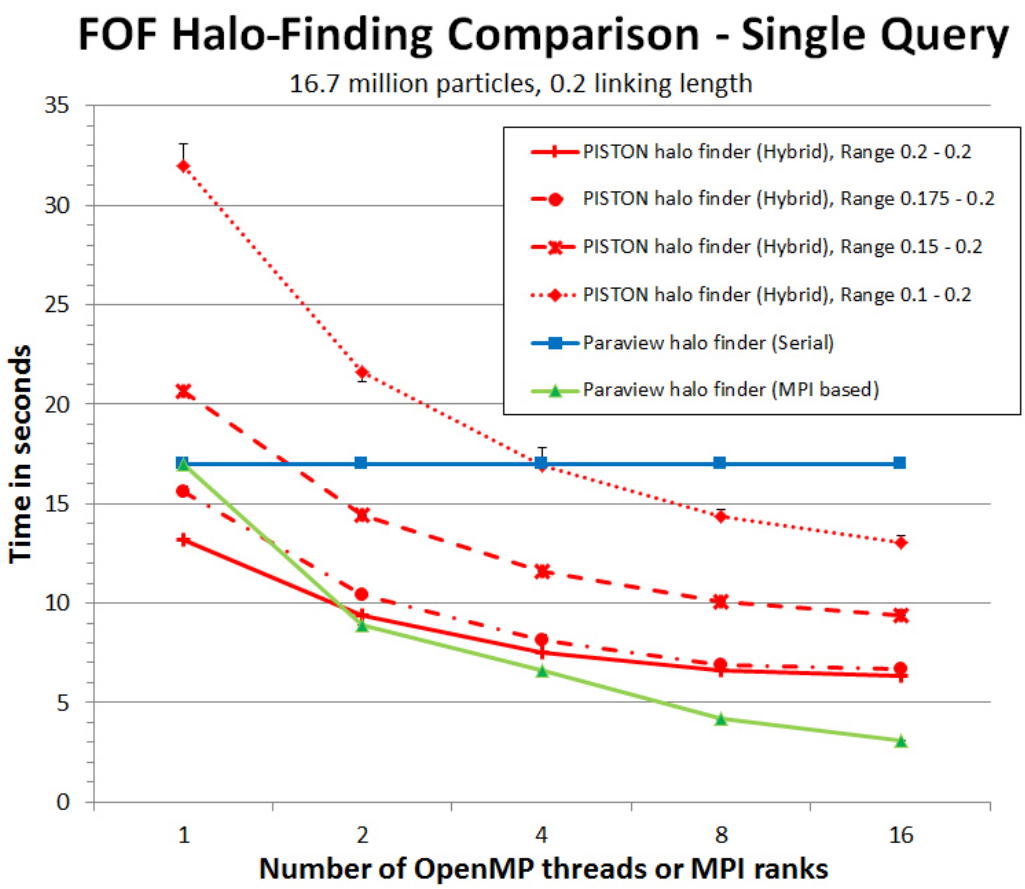


Figure 3.7: Comparing timing results of PISTON-based hybrid halo-finding algorithm, for a single query, against the Paraview implementation run serially and with multiple MPI processes.

the increased run-times for longer linking length ranges in Figure 3.7.

Nevertheless, when performing multiple linking length queries, the proposed feature hierarchy clearly has an advantage over the Paraview halo finder even when run with multiple MPI processes, see Figure 3.8. The feature hierarchy needs to be computed once, and then only a traversal within the hierarchy is needed, which takes very little time relative to the construction time, to obtain halo information for multiple linking lengths queries. For the Paraview implementation, a full re-run of the algorithm is needed for every linking length change. Therefore, depending on the number of cores, it becomes more efficient to use the feature hierarchy when more than a minimum of two to five queries are performed.

Furthermore, a typical FOF halo-finding algorithm constructs halos by making use

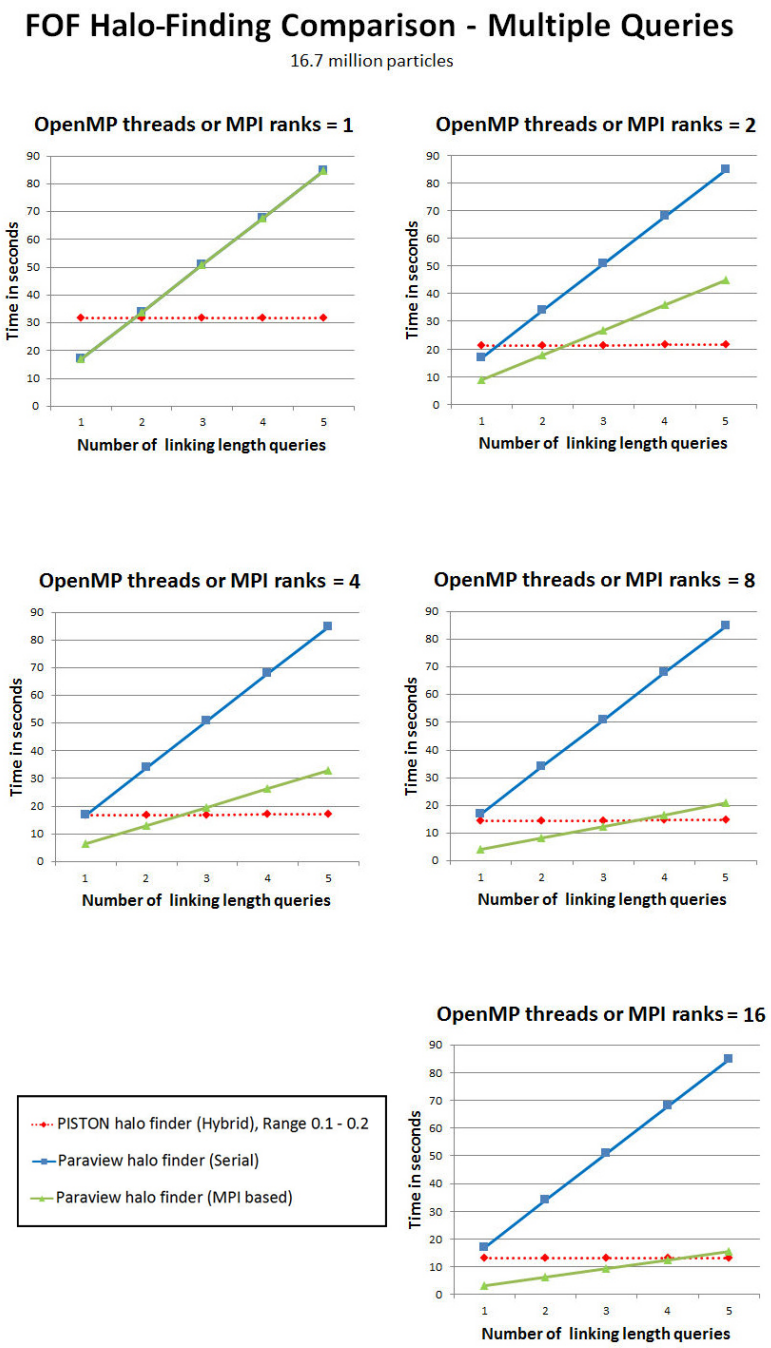


Figure 3.8: Comparing timing results of the PISTON-based hybrid halo-finding algorithm, for subsequent queries, against the Paraview implementation run serially and with multiple MPI processes. At each query, Paraview implementation always computes halo details for just one linking length, whereas the algorithm used in this dissertation produces a hierarchy with 100 discretized linking length values for the range from 0.1 to 0.2. The hierarchy is created during the first query and is used throughout the rest of the queries.

of two parameters: linking length and halo size. First, an extended neighbor search is performed based on the linking length parameter specified. After connecting all pairs of particles that lie closer than the specified linking length, the FOF halo-finding algorithm results in a network of linked particles. Each connected component found in this network is considered as a single FOF halo. Finally, the discovered halos are filtered based on the halo size parameter, where all the halos with fewer particles than the specified value are ignored. In this setting, whereas the feature hierarchy enables interactive exploration of halos for a range of linking lengths, the feature-based attributes stored within the hierarchy, particularly the halo size parameter, provide the additional flexibility to interactively filter the extracted halos by their size. For this particular cosmology data set, Figure 3.9 shows an example of a set of halos extracted from one of the halo hierarchies created. More results related to this application example can be found in [162].

3.2.2 Combustion Data - Hydrogen Flame with No Turbulence

Here, a combustion data set from an AMR simulation of an idealized premixed hydrogen flame with no turbulence is considered [165]. It contains 100 time steps at an effective resolution of $256 \times 256 \times 768$ and totals around 400GB. In this data set, the feature of interest is a burning cell defined as a region of high fuel consumption. According to the collaborating scientists, as there exists no unique fuel consumption value, exploring the burning cells under varying fuel consumption values is of significant interest. As a result, a nested disjoint feature hierarchy is used to encode the feature families for burning cells.

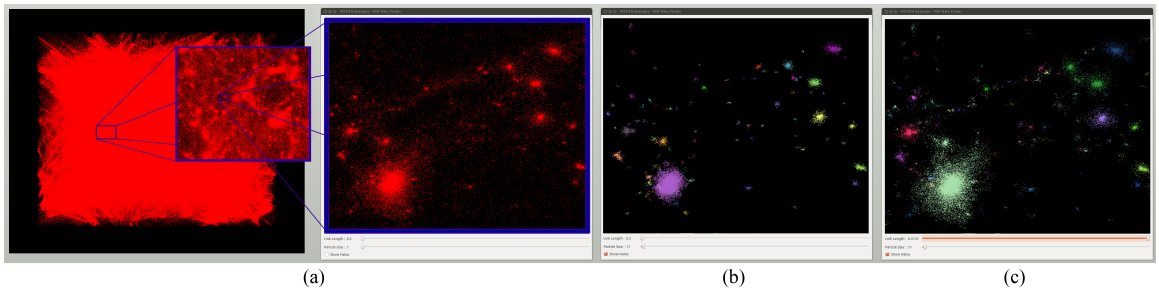


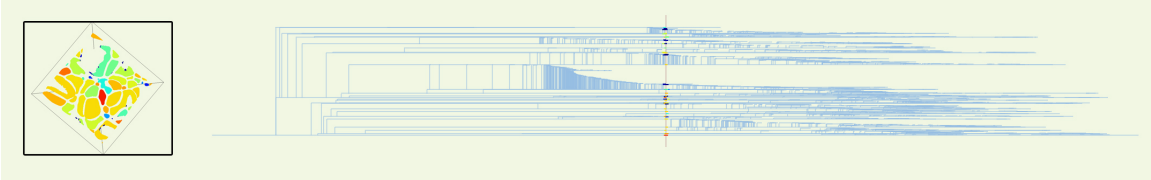
Figure 3.9: Halos found on a set of particles from a cosmological simulation: (a) Data set contains 16.7 million dark matter particles. (b) Halos with linking length 0.2 and halo size 11. (c) Halos after increasing the linking length to 0.4141. Each set of particles in the same color in (b) and (c) represents a single halo.

In order to obtain the necessary feature hierarchies, a set of one-time preprocessing steps is performed offline for each time step in the data set in an embarrassingly parallel fashion. Again, due to the large data sizes within the original data set (400GB), these preprocessing steps have been performed at NERSC on the Carver cluster with 16, 2.67 GHz Intel Xeon processors per node [166]. Here, the clustering hierarchy of these burning cells is computed using threshold-based segmentation, and the resulting data consist of the feature hierarchies as well as the corresponding segmentation and feature-based attribute information. In this case, various feature-based attributes such as feature volume, average temperature, average H_2 consumption rate, and position are computed and stored within the hierarchy. After the preprocessing steps to create the feature hierarchies, the data size is reduced to ≈760MB.

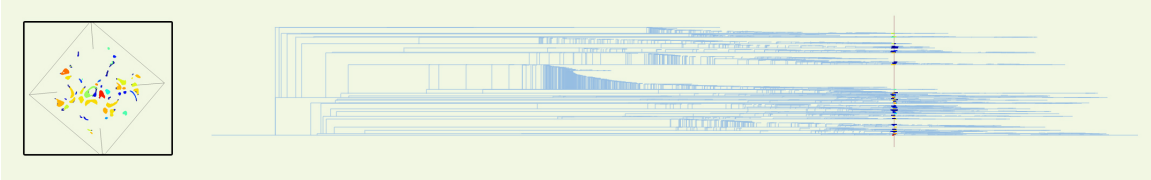
For one time step of this data set, Figure 3.10 shows the resulting features for varying fuel consumption rates. The results show that as the fuel consumption rate increases, burning cells decrease in both number and size, with only stable burning cells remaining. Furthermore, an example in which localized, per-feature parameter values are defined with the use of arbitrary cuts is shown in Figure 3.11. According to the collaborating scientists, having this type of flexibility to define and extract features using fixed and arbitrary cuts and also the ability to explore the entire feature space is extremely useful in understanding combustion data. More results related to this particular application example can be found in [125], [167].

3.2.3 Image Data

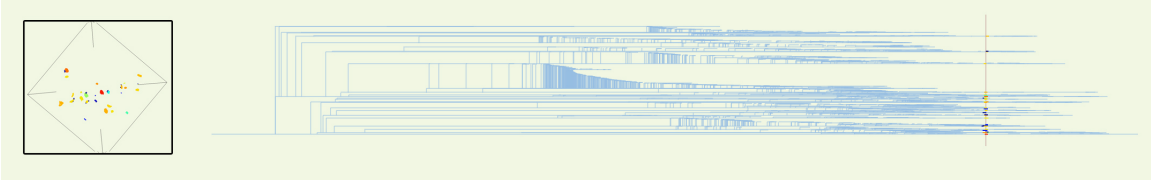
Finally, the Berkeley Segmentation Data Set 500 (BSDS500) is considered [168]. This data set provides an empirical basis for research on image segmentation and boundary detection. It consists of 500 natural images that have been manually segmented and total about 36MB. As per the feedback from the collaborating scientists, use of a nested disjoint feature hierarchy that represents the hierarchy of region merging provides much needed flexibility to explore the image segmentation details, and in turn assist in extracting better image segmentation results. For this data set, the supervised hierarchical approach by Liu et al. [169], [170] is used to compute the clustering hierarchy of image regions for a range of region saliency values. The resulting feature hierarchies total about 12MB. Figure 3.12



(a) Fuel consumption rate = 10



(b) Fuel consumption rate = 15



(c) Fuel consumption rate = 17

Figure 3.10: Effects of varying the fuel consumption rate to explore the entire feature space. Here, for a particular time step, the resulting burning cells (in left) and their feature hierarchy (in right) are shown at (a) 10, (b) 15, and (c) 17 fuel consumption rates. In each case, the horizontal graph layout of the feature hierarchy is displayed, and the cut within the feature hierarchy is indicated with a red vertical line.

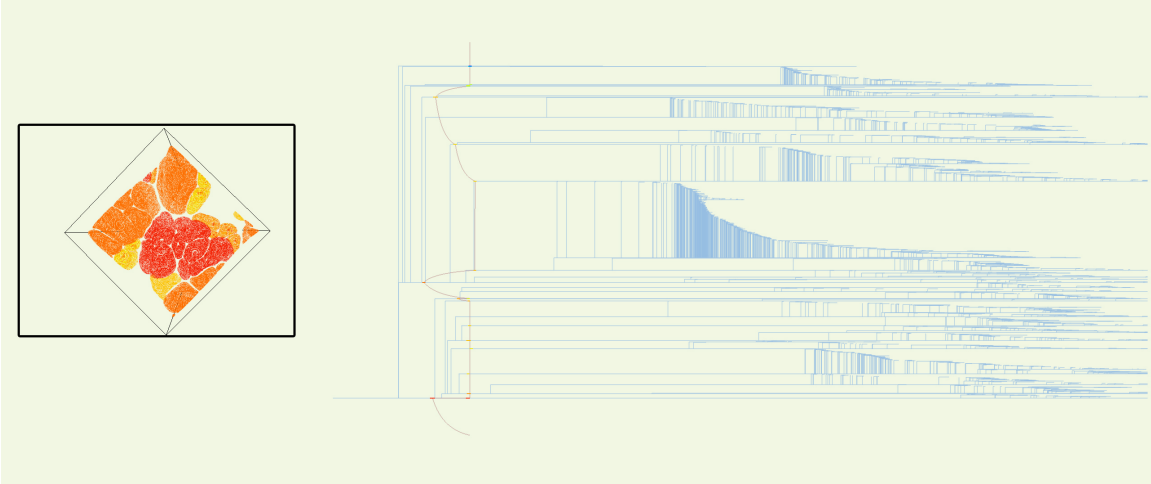
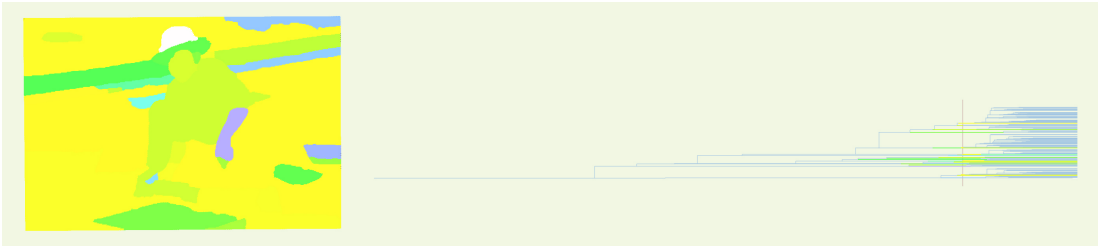
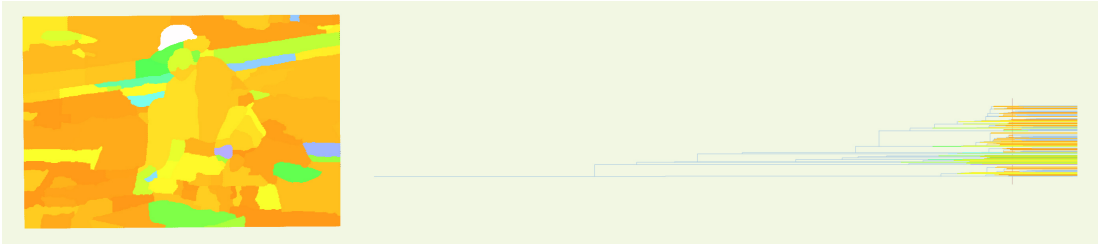


Figure 3.11: An arbitrary cut (indicated with a red line within the feature hierarchy) that results in localized, per-feature parameter values, and its resulting features (in left) are shown.

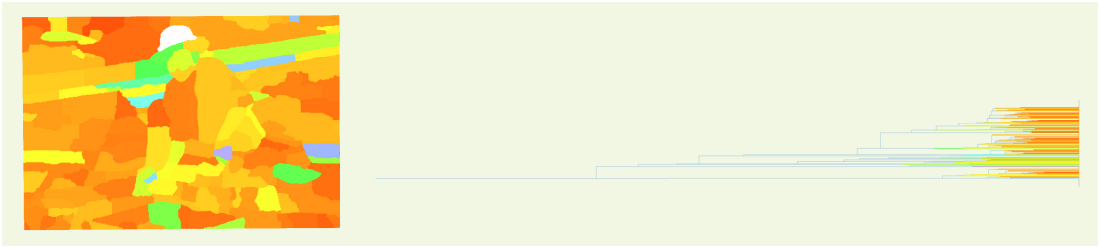
(a) Original image



(b) Region saliency value = - 0.16



(c) Region saliency value = - 0.09



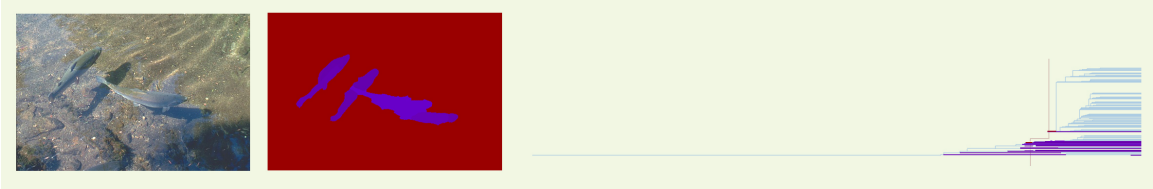
(d) Region saliency value = 0

Figure 3.12: Effects of varying the region saliency value to explore the entire feature space. For the original image shown in (a), its results are shown at (b) - 0.16, (c) - 0.09, and (d) 0 region saliency values. In each result, the image segmentation result (left), the horizontal graph layout of the feature hierarchy (right), and the cut within the feature hierarchy (indicated with a red vertical line) are displayed.

shows the effects of varying the region saliency value to explore the entire feature space. For this type of image data, the capability within the feature hierarchy to define localized, per-feature parameter values is particularly useful. In fact, it enables flexible image segmentation within an interactive setting. Figure 3.13 presents an example of localized, per-feature parameter values used to extract more intuitive image segmentation results.

3.3 Summary

This chapter discusses how interactive feature exploration can be achieved for univariate features via a hierarchical feature representation. Specifically, a feature hierarchy that encodes the clustering hierarchy of features for a range of parameter values is presented. Depending on the type of hierarchy, in addition to the feature hierarchy details, feature segmentation details and various feature-based attribute details are also stored within this feature representation. As a result, once the hierarchy is created, it has the capability to interactively explore features, along with their segmentation, and other feature-based attributes for their entire parameter range in a more flexible manner. Furthermore, the abstract nature within the feature hierarchy enables it to be used across a wide variety of application domains. This chapter presents hierarchy construction, feature extraction, and other implementation details related to this feature representation and also discusses



(a)

(b)

Figure 3.13: Two examples where arbitrary cuts are used to obtain more flexible and intuitive image segmentation results. In each example, the original image (left), image segmentation result (middle), the horizontal graph layout of the feature hierarchy (right), and the arbitrary cut within the feature hierarchy (indicated with a red vertical line) are displayed.

its advantages and limitations. Finally, several application examples are presented to demonstrate the utility of this hierarchical feature representation.

CHAPTER 4

COMPUTING AND ENCODING BIVARIATE FEATURES AT MULTIPLE SCALES

With the deluge of data from experiments and simulations, multivariate dynamic data sets are becoming increasingly common. Application of feature-based analysis to multivariate data requires investigating complex interactions and relationships associated with different variables in the data set. However, as such application involves the exploration of a number of variables, techniques intended for univariate features are not sufficient to effectively analyze these multivariate data sets, thus calling for new alternatives to existing univariate feature-based approaches. Consider, for example, halos in cosmology simulations: Halos are typically defined via the distance between particles, called the linking length. Yet, as two halos cross paths, a distance-based approach cannot distinguish one from the other, and thus provides incorrect results. Instead, simultaneously looking at the velocity vector of the particles would likely allow separation of such entangled halos.

This chapter focuses on addressing the problem of interactive feature exploration for bivariate features. Specifically, with a focus on *disjoint nested feature hierarchies*, which is a hierarchy in which every feature is nested and has only one parent, a simple yet effective approach to combine two univariate feature hierarchies into a single bivariate one is introduced. The result is a *bivariate graph*, a novel compact representation of all bivariate features defined by any combination of the two parameters. This, for the first time, allows users to freely explore two parameter feature definitions in large-scale data. Here, in order to maintain its applicability across a range of application domains, appropriate abstractions are maintained within the feature representation. This chapter provides comprehensive details about this feature representation and presents several application results to demonstrate its utility.

4.1 Bivariate Graphs

The ability to freely change any of the parameters provides significantly more flexibility in the parameter exploration, especially for multivariate data sets. However, with few exceptions as discussed in Section 2.1.2, existing feature-based approaches do not support multiple parameters. Instead, such cases are typically tackled by defining some metric on the joint parameter space, combining all values to a single scalar, which is then used during the analysis. Common examples of this approach are the various distance metrics proposed for multivariate clustering [85–88]. However, often this joint value is unintuitive to use, and the metrics are chosen ad hoc. More importantly, such a representation no longer allows users to explore each parameter independently.

This chapter introduces a general approach to combine two univariate families of features into a single bivariate representation that allows a direct and interactive manipulation of both input parameters. In particular, with a focus on disjoint nested feature hierarchies, an approach to construct a novel bivariate feature representation is presented. More specifically, two types of disjoint nested feature hierarchies are considered: partitioning and subsetting. *Partitioning hierarchies* are disjoint nested hierarchies for which the union of all features at any parameter value partitions the entire data set, for example, a hierarchical clustering. *Subsetting hierarchies* are disjoint nested hierarchies for which the union of all features covers only a subset of the data, for example, a merge tree. Both types of feature hierarchies can be described using a tree whose nodes represent the features in the data. Here, new algorithms that combine two such hierarchies storing univariate features into a single bivariate graph are introduced. Specifically, the resulting bivariate graph is in the form of an *overlapping nested feature hierarchy* where features are nested and at least one feature contains multiple parents.

This proposed approach is complementary to existing techniques for bivariate feature exploration, and applies directly to a wide range of hierarchies, from generic clustering [70], to watersheds [171], merge trees [14], or Morse complexes [63]. Once created, a bivariate graph enables users to interactively extract and visualize bivariate features at any parameter combination, thereby providing a flexible exploration and analysis platform. Due to its abstract nature, the abstract aspects should first be defined to create this type of a hierarchical feature representation. Specifically, as the proposed algorithms combine two

univariate feature hierarchies to form a bivariate hierarchy, the relevant abstract details for each univariate feature hierarchy – the feature of interest and method of grouping for each univariate hierarchy – should be specified.

4.1.1 Efficient Representation of a Bivariate Hierarchy

The goal in representing a bivariate hierarchy is twofold. First, such a representation must be able to efficiently extract bivariate features at any arbitrary parameter combination; and second, the representation must be sufficiently compact to be practical for large-scale simulations. Similar to the canonical trees of univariate hierarchies, bivariate features result in a graph representation, specifically in an overlapping hierarchy. Each node in the graph represents a (simply connected) feature at a specific parameter combination, and the branches represent connections to features at neighboring parameter values, see Figure 4.1.

The naive approach to construct this bivariate graph is to simply precompute all bivariate features for all parameter combinations. However, without taking advantage of the inherent nesting relationships, this approach results in a graph that is often unmanage-

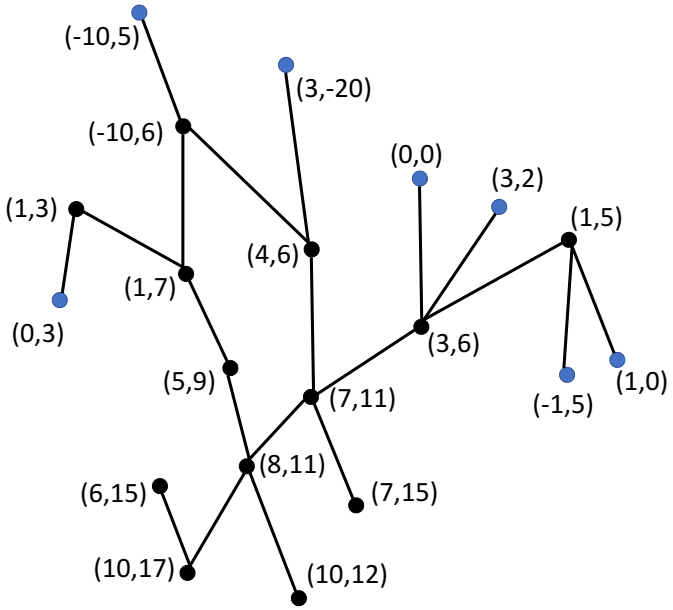


Figure 4.1: An example bivariate graph. Here, each node represents a feature at a specific parameter combination, and its branches represent connections to features at neighboring parameter values. The leaf nodes within the hierarchy are indicated in blue, and the parameter combination at each node is also indicated.

ably large and complex. Instead, this chapter focuses on computing a smaller and more compact graph with a reduced set of edges while still allowing the full exploration of the entire parameter space.

A univariate disjoint nested feature hierarchy is able to efficiently encode the clustering hierarchy of univariate features for its entire parameter range. For the bivariate case, joint consideration of their corresponding univariate disjoint nested feature hierarchies is adequate to present the required bivariate feature information. However, such an approach requires additional computation, both to maintain two univariate feature hierarchies and to combine their results for obtaining the necessary bivariate features, which is not optimal. Therefore, this dissertation focuses on combining two univariate disjoint nested feature hierarchies into a single bivariate feature representation that can efficiently encode the hierarchical nature within bivariate features.

Here, it is important to note that the resultant bivariate feature hierarchy differs from univariate disjoint nested feature hierarchies in several respects. First, a node within the bivariate hierarchy can have multiple parents instead of just one as in univariate disjoint nested feature hierarchies. Consequently, even though both univariate disjoint nested feature hierarchies can be represented using trees, their bivariate hierarchy can be represented only by a graph. Second, for these two cases, the feature representation at a particular cut is different. As discussed in Section 3.1.2, univariate disjoint nested feature hierarchies create a forest at a particular cut, where each connected component in the forest, a subtree, represents a feature existing at that cut. Specifically, for a univariate disjoint nested feature hierarchy, the connected component details are directly encoded within the subtree and can be extracted by looking at that subtree's *active* branch that crosses the cut. For bivariate graphs, this cut results in a subgraph with multiple roots. Even though each connected component in the subgraph represents a feature existing at the cut, this subgraph no longer directly encodes those connected component details. Therefore, at a particular cut, a graph traversal within the resulting subgraph (towards its leafs) is needed to identify the features defined by the cut, see Figure 4.2. Finally, the lifetime of a feature in a univariate disjoint nested feature hierarchy is slightly different from the lifetime of a bivariate feature. In the case of a univariate disjoint nested feature hierarchy, the lifetime of a feature is indicated by the length of its branch – the feature is born at the parameter value of the branch's

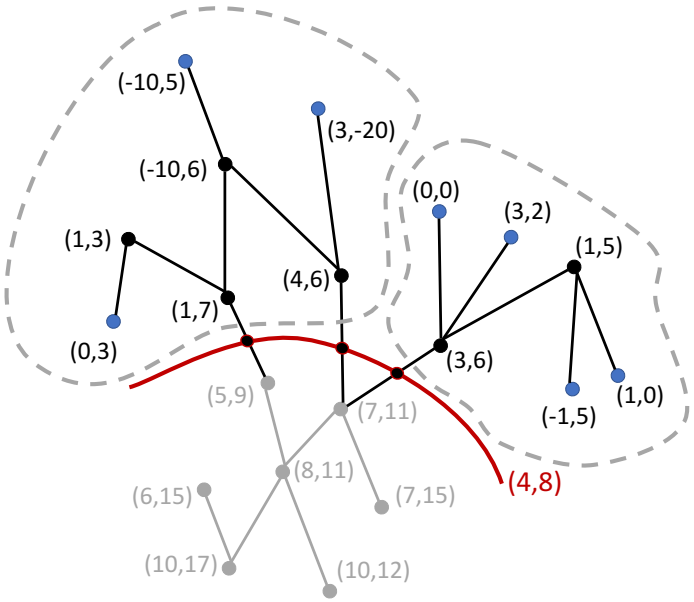


Figure 4.2: For the example bivariate graph in Figure 4.1, a cut (indicated in red) at the parameter combination (4,8) and its resultant features (outlined in dashed lines). Here, the cut results in a subgraph, and each connected component within the subgraph represents a feature existing at that cut.

top node and is active until it merges with its parent at the parameter value of its bottom node. However, as a bivariate feature has multiple parents, determining when it should be considered merged is not straightforward. Therefore, instead of looking at the lifetime of a feature as in univariate disjoint nested feature hierarchies, in bivariate hierarchies the birth time of each feature is considered.

4.1.2 Graph Construction

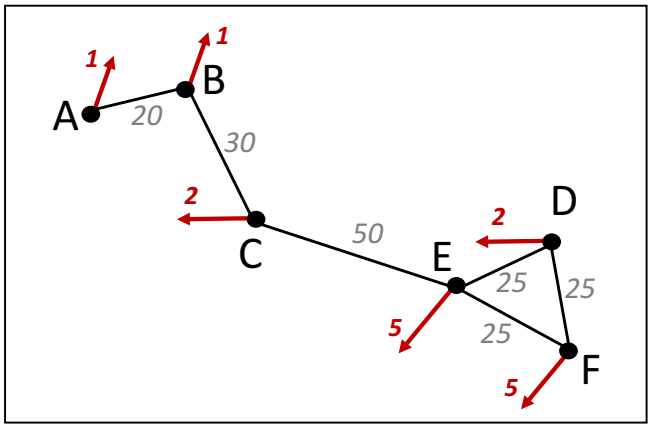
This section presents a general and flexible approach that combines two univariate disjoint nested feature hierarchies into a single bivariate feature representation. In particular, for any two partitioning or two subsetting feature hierarchies, two algorithms to construct their equivalent bivariate feature hierarchy are discussed.

4.1.2.1 Combining Partitioning Feature Hierarchies

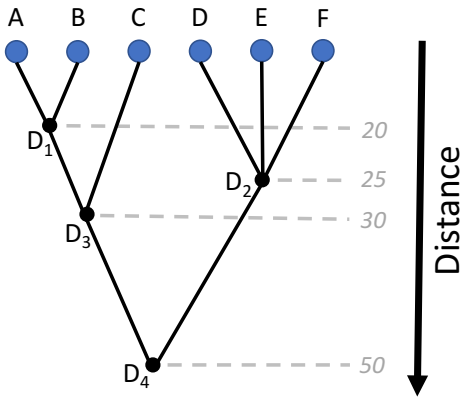
The algorithm to construct a bivariate feature hierarchy from two partitioning hierarchies proceeds in three steps. Given two partitioning feature hierarchies, the first step toward constructing their bivariate graph is computing the resulting bivariate graph's leaf

nodes. In the examples, it is assumed that both hierarchies start from the same leaf nodes (as in a clustering). If this is not the case, leafs are created by pairwise intersection of the leafs in both hierarchies. Next, the remaining structure of the bivariate graph is computed by combining the information in the two partitioning feature hierarchies. Finally, the resulting graph is simplified to remove redundant nodes.

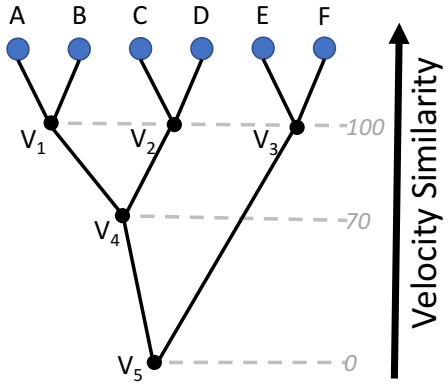
Consider the halo example in Figure 4.3. Particles $A - F$ are clustered based on their pair-wise distance, creating hierarchy h_1, as well as their similarity in velocity (for example the dot product between their velocities) creating hierarchy h_2, as shown Figure 4.3b and 4.3c, respectively. After the leaf nodes are identified, the rest of the bivariate hierarchy



(a)



(b) Distance hierarchy



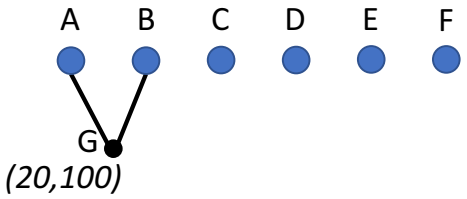
(c) Velocity similarity hierarchy

Figure 4.3: Particles shown in (a) are clustered based on their distance and velocity similarity to produce the partitioning feature hierarchies in (b) and (c). Here, (a) shows a particle's velocity information in red and pair-wise distances in black.

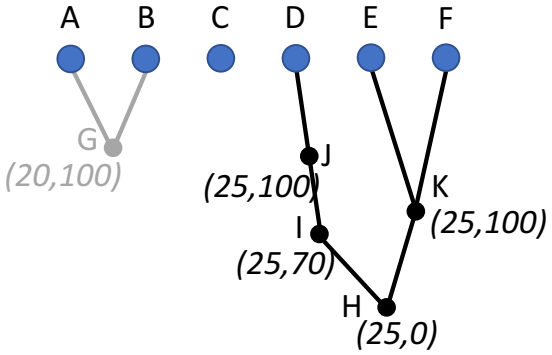
is created by combining h_1 and h_2. Specifically, all nonleaf nodes in hierarchy h_1 are considered in order of the increasing 'height' of the tree, in this example increasing distance (i.e., D_1, D_2, D_3, and D_4). For each node, its least common ancestor (LCA) in the hierarchy h_2 is obtained. The LCA of a node X is defined as the first node in the hierarchy h_2 that contains all elements of X. For example, the LCA nodes in the velocity hierarchy of D_1, D_2, D_3, and $D4$ are V_1, V_5, V_4, and V_5, respectively. Intuitively, the subtree rooted at the LCA defines how the features in h_1 may be further split by the second hierarchy h_2. In order to encode this information in the bivariate graph, the subtree of the LCA is traversed in a depth-first fashion, adding nodes as necessary to the bivariate hierarchy. During this traversal, each node in the subtree is checked against the existing nodes within the bivariate hierarchy. If a node already exists, its corresponding subtree has already been created, so the recursion is stopped and the next node is processed. Here, to quickly check whether a node already exists, a checksum is maintained of all nodes in hierarchy h_2 and the bivariate hierarchy. This checksum is created from the ids of the nodes. When traversing an LCA's subtree, the checksum of each node in the subtree is checked against the checksum of the current nodes within the bivariate hierarchy.

The resulting bivariate graph for each node in h_1 at the end of this step is presented in Figure 4.4. First, node D_1 is considered, and the subtree of its LCA, V_1, is traversed in depth-first order to consider V_1, A, and B nodes, see Figure 4.4a. When traversing the subtree, node V_1 is first considered. As V_1 does not exist in the bivariate graph, a node G at $(f = 20, g = 100)$ is added to the bivariate graph. Here, the function values for both parameters are obtained by looking at the node considered in hierarchy h_1 (which is D_1 at $f = 20$) and the current node within its LCA's subtree (which is V_1 at $g = 100$). Then, the next node within the subtree, node A, is considered. As A already exists within the bivariate graph, nodes G and A within the bivariate graph are connected. Next, the final node in the subtree, node B, is considered. As node B also exists within the bivariate graph, nodes G and B are connected. Likewise, the rest of the nodes in hierarchy h_1 are processed.

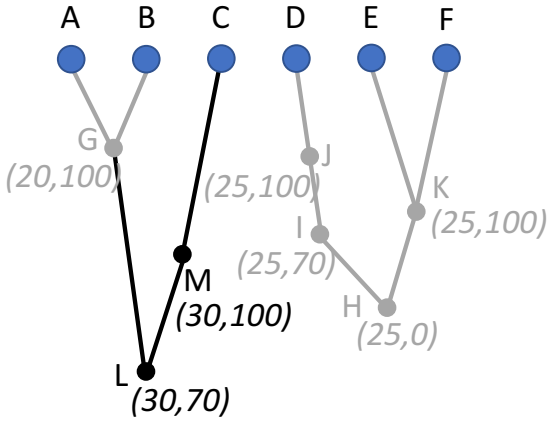
Note that, for any node X in h_1, when its LCA's subtree is traversed, only nodes that contain descendants of X are considered. For example, consider node D_2 in h_1, and its LCA V_5, see Figure 4.4b. The node D_2 contains D, E, and F elements. Therefore, when the subtree of V_5 is traversed, only nodes V_5, V_4, V_2, D, V_3, E, and F are considered. Node $V1$



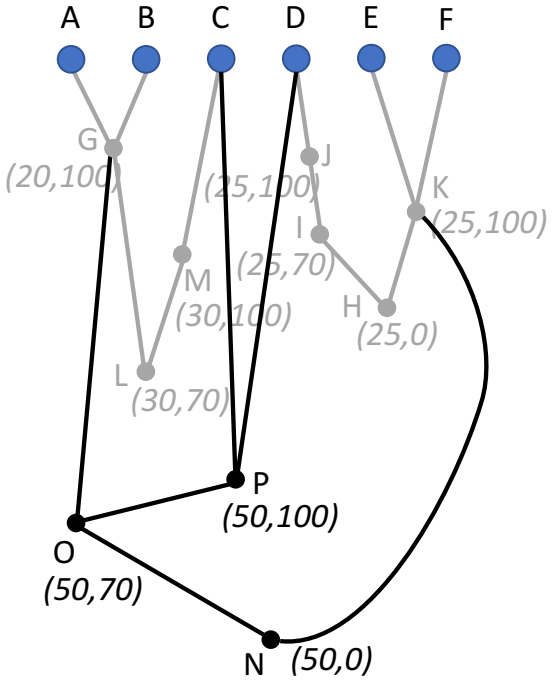
(a) Considering D_1 and its LCA V_1



(b) Considering D_2 and its LCA V_5



(c) Considering D_3 and its LCA V_4



(d) Considering D_4 and its LCA V_5

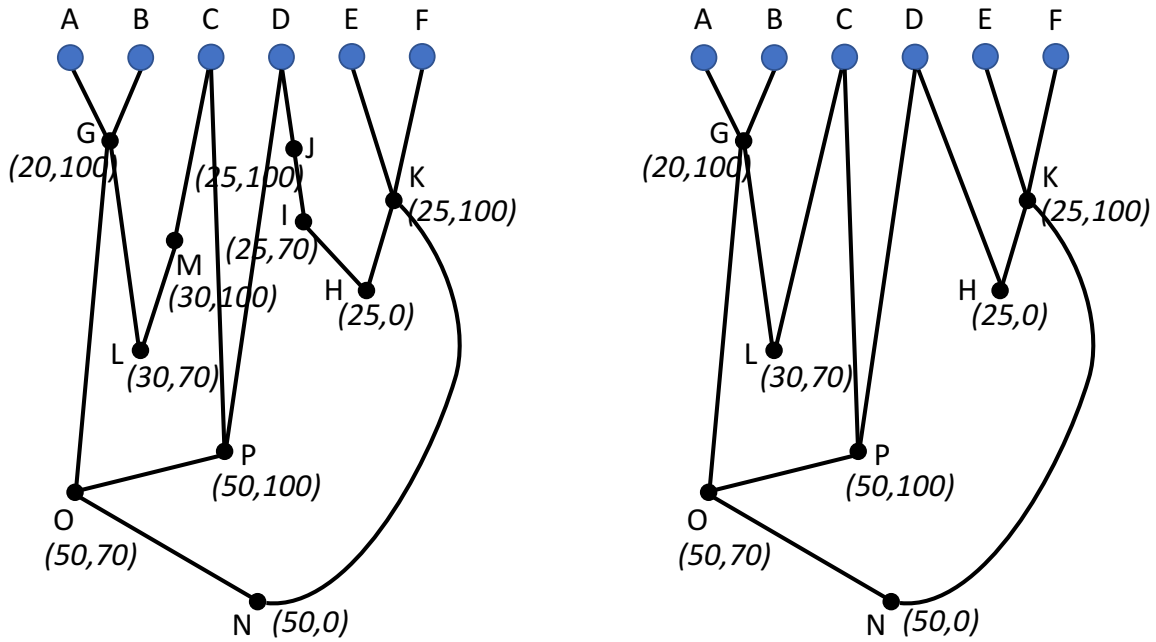
Figure 4.4: Nodes within the distance hierarchy are considered according to the increasing order of level. For each node, its LCA in the velocity hierarchy is found and the LCA's subtree is traversed to create the bivariate hierarchy. The resulting bivariate graph at the end of each step is shown in (a) – (d).

and all its descendants are skipped as they contain no descendant of D_2. Node C is also skipped for the same reason. For this example, the resulting bivariate graph after all nodes in h_1 have been processed is shown in Figure 4.5a. In a final third step, the resulting graph is further simplified by removing its redundant valence two nodes to result in the graph in Figure 4.5b.

4.1.2.2 Combining Subsetting Feature Hierarchies

As discussed above, subsetting feature hierarchies define features as subsets of the entire domain. As a result, each node within this hierarchy contains exclusive elements. This fact is used to create a slightly different algorithm to combine two subsetting feature hierarchies into their equivalent bivariate graph.

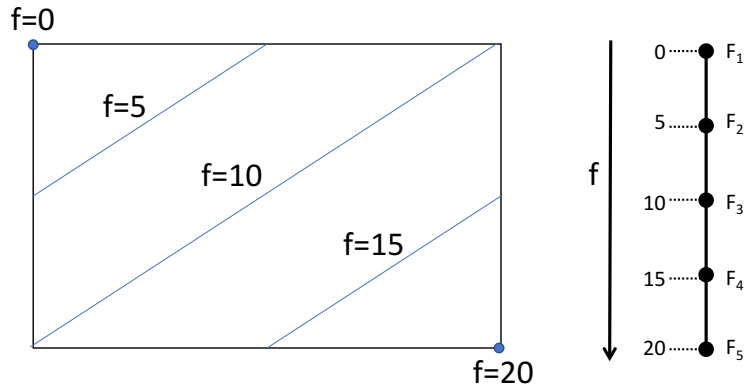
Given two subsetting feature hierarchies for parameters f and g, the algorithm proceeds in three steps. First, for each node in both hierarchies, its subnodes are obtained by looking at its exclusive elements in the other hierarchy. Consider the example in Figure 4.6. The node F_2 in hierarchy f exists at parameter value $f = 5$. In hierarchy of g, its exclusive



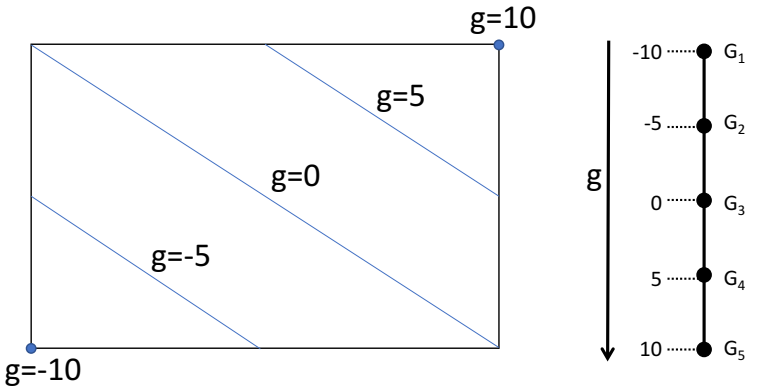
(a) Bivariate graph

(b) Simplified bivariate graph

Figure 4.5: For the two hierarchies in Figure 4.3, the resulting bivariate graph is displayed in (a). This graph can be further simplified by reducing its valence two nodes to produce the graph in (b).



(a) Hierarchy of f

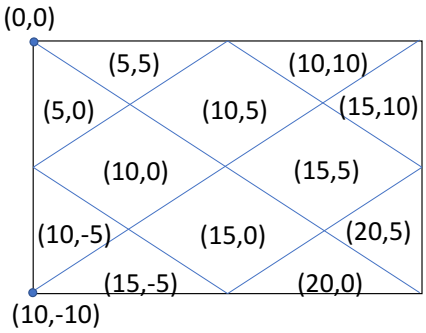


(b) Hierarchy of g

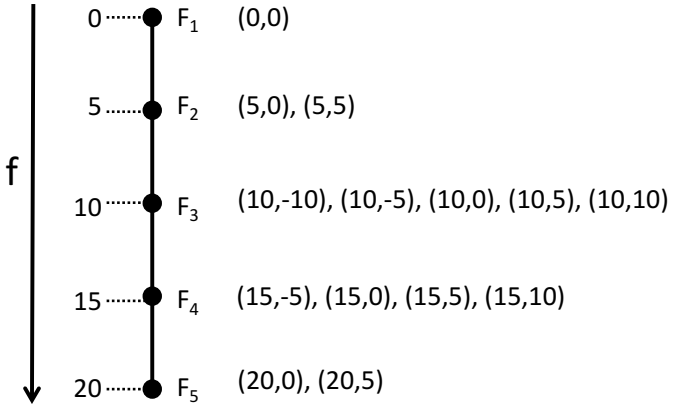
Figure 4.6: For the same data domain, the subsetting feature hierarchies for parameters f and g are shown in (a) and (b).

elements can be subdivided into two regions with parameter values $g = 0$ and $g = 5$. As a result, F_2 can be subdivided into two subnodes, each existing at ($f = 5$, $g = 0$) and ($f = 5$, $g = 5$). Likewise, for every node in both hierarchies, its subnodes can be obtained. For the example in Figure 4.6, the extracted subnodes for both hierarchies are presented in Figure 4.7. Then, the intersection of these subnodes is added to the bivariate graph.

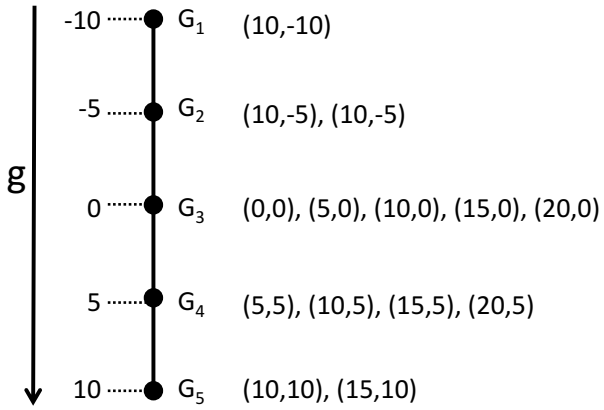
In a second step, for each node in both subsetting hierarchies, the edges existing across its subnodes are obtained by considering the subnodes' neighboring information in the domain (as shown in Figure 4.7a). Then, those edges are added to the bivariate graph. For the node F_2 in hierarchy of f, its subnodes ($f = 5$, $g = 0$) and ($f = 5$, $g = 5$) contain an edge as they are neighboring regions. For the example in Figure 4.6, the extracted edges for both hierarchies are presented in Figure 4.8. At the end of these three steps, all the nodes and edges produced make up the final bivariate graph. Again, in a final third step,



(a)



(b) Sub nodes for hierarchy of f



(c) Sub nodes for hierarchy of g

Figure 4.7: For each node in a hierarchy, its subnodes can be found by looking at its exclusive nodes in the other hierarchy. Subnodes found for each node in both hierarchies f and g are shown in (b) and (c).

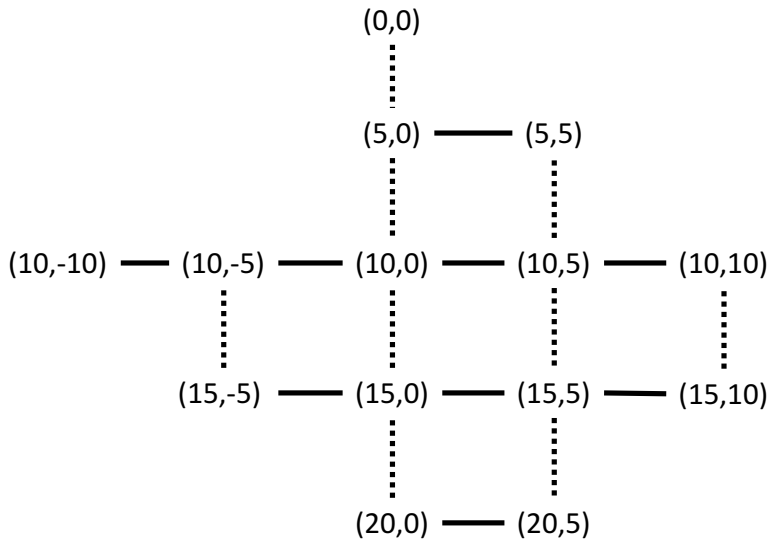


Figure 4.8: For each node in both hierarchies, the edges across its subnodes are computed by looking at their neighboring in the domain as shown in Figure 4.7a. The resulting edges for hierarchy f are shown in black solid lines, and the edges for hierarchy g are shown in black dashed lines.

the resultant bivariate graph can be further simplified by removing its valence two nodes. Note that whereas in this simple example the final graph describes the tensor product of the two input hierarchies, and thus contains a square number of nodes, in practice this is unlikely to occur. Usually, feature parameters are related (rather than orthogonal as in this example), and thus far fewer combined features are typically found.

4.1.3 Feature Extraction

Bivariate hierarchies store the entire family of bivariate features in a compact graph structure. As a result, once constructed, this structure has the ability to easily extract bivariate features for any parameter combination. For example, given a parameter combination (f_i, g_j), the corresponding features are extracted from the bivariate hierarchy by 'cutting' it at (f_i, g_j). For the bivariate graph in Figure 4.8, its resultant features for the cut at $(f = 10, g = 5)$ are shown in Figure 4.9. As previously discussed, a bivariate hierarchy is represented using an overlapping hierarchy, and therefore, this cut results in a subgraph with multiple roots. From this cut, the subgraph is traversed 'upwards' to extract all active features. During this graph traversal, the connected components within

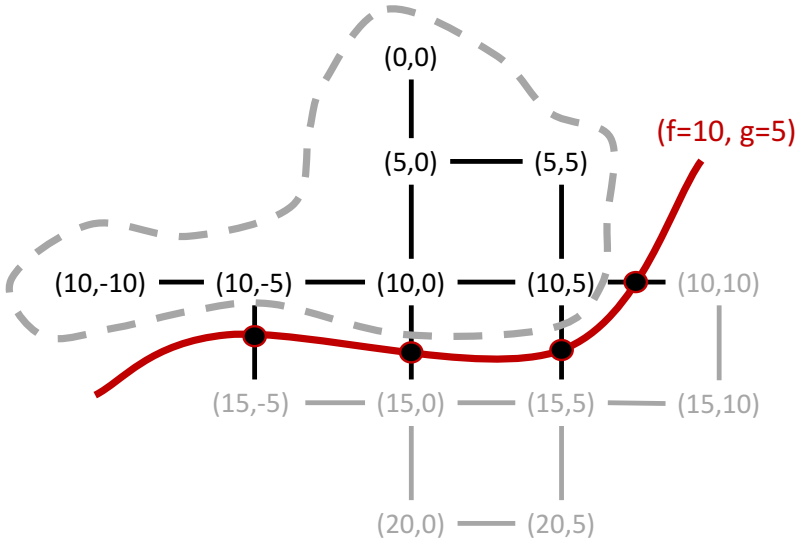


Figure 4.9: For the bivariate graph in Figure 4.8, its cut at $(f = 10, g = 5)$. This cut results in a subgraph where each connected component represents a feature existing at $(f = 10, g = 5)$. Here, the cut is indicated in red, and its feature is outlined in a dashed line.

the resulting subgraph are computed, each representing a feature existing at (f_i, g_j). Along with the features, the feature-based attributes and geometry details are also computed on a per-connected component basis.

4.1.4 Implementation Details

Given a data set, an offline preprocessing step is used to compute the univariate disjoint nested feature hierarchies of interest using any one of the existing algorithms. Both hierarchies are encoded as trees, which are subsequently combined into a bivariate hierarchy and stored in a file. Just as discussed in Section 3.1.3, within this file, for each feature in a hierarchy, pointers to its parents and descendants, corresponding parameter values (for all its parent branches), exclusive elements, and feature-based attributes are stored.

At run time, these details are read from the file and stored in a range tree data structure [156]. A range tree is an ordered tree data structure that stores a set of points and allows quick retrieval of points within a given range. In the case of bivariate hierarchies, all features are stored according to their birth time in a 2D range tree to quickly extract all features that are alive at a given parameter combination. An additional look-up structure is also maintained to store the hierarchical structure of the bivariate graph. Within

this look-up structure, the parameter values, parent details, and segmentation details are stored for each feature. Together, these two data structures allow interactive exploration of bivariate features for their entire parameter range.

For a given parameter combination (f_i, g_j), the query $((-\infty, f_i], (-\infty, g_i])$ is considered, and the range tree is processed to return all features that are alive. In a subsequent step, those nodes are traversed using the graph to extract their connected components. Each connected component represents a feature existing at (f_i, g_j). During this traversal, the feature-based attributes and segmentation details of each feature are computed on a per-connected component basis.

4.1.5 Advantages and Limitations

This chapter presents a general approach to combine two, one-parameter families of features into a single bivariate representation that efficiently encodes the nesting relationships among bivariate features. In particular, it compactly represents bivariate hierarchies by using a reduced set of edges while still allowing full exploration of the parameter space. As a result, unlike existing approaches, this bivariate feature representation allows a direct and interactive manipulation of both input parameters. Once created, the bivariate hierarchy can be interactively traversed to extract features by proper selection of connected components. Therefore, with the use of this proposed feature representation, scientists can freely explore two parameter feature definitions in large-scale data.

One limitation is that the proposed bivariate graph construction approach is restricted to univariate disjoint nested feature hierarchies, more specifically to only two types of disjoint nested feature hierarchies, partitioning and subsetting. Furthermore, in the worst case, the resulting bivariate graph can be the tensor product of the two input hierarchies; however, in practice that is highly unlikely to happen. Often, two correlating parameters are used, resulting in fewer combined features within the resulting bivariate graph.

4.2 Application Results

This section presents the results of two real-word data sets: one using partitioning hierarchies to improve halo detection and one using subsetting hierarchies to analyze atmospheric science simulations.

4.2.1 Cosmology Data

Cosmology simulations involve a large set of dark matter particles to observe how halos form. In this context, a halo is defined as an over-dense region of particles [157], [158]. The most common definition of a halo is based on a friends-of-friends (FOF) clustering [159], where all particles that are reachable through links shorter than a predefined distance, called the linking length, are considered to be in one halo. However, as halos can cross paths over time, a distance-based approach, such as FOF, is not sufficient to separate entangled halos. Instead, looking at both distance and velocity of particles enables a more accurate identification of halos.

Here, a data set containing 16.7 million particles from a 256^3 cosmological simulation is considered (the same as considered in Section 3.2.1), and a bivariate hierarchy based on distance and velocity similarity is constructed. To compare velocities, the cosine of their angle multiplied by the difference in magnitude is used as a similarity metric. First, the univariate feature hierarchies for both distance and velocity similarity are constructed for a particular time step of the data set. Here, all hierarchies are constructed for realistic parameter values. The longest linking length value cosmologists typically consider is 0.2, and therefore, a linking length range around this value (e.g., 0.1 - 0.21) is selected for the distance hierarchy. When creating the velocity hierarchy, the fact that particles in different distance clusters for linking length 0.21 will never be considered in the same halo is exploited. Consequently, velocity hierarchies restricted to the largest distance-based halos are constructed to avoid unnecessary computation.

Once the univariate disjoint nested feature hierarchies are constructed, they are combined using the proposed algorithm for partitioning hierarchies. For one specific time step of this data set, its raw data totals about $76GB$; each of the univariate disjoint nested feature hierarchies, distance and velocity, totals $2MB$ and $537MB$, respectively; and the bivariate hierarchy is about $9MB$. File sizes for the other time steps of the data set exhibit a similar pattern. These results show that it is more efficient to keep one feature representation for bivariate feature exploration rather than maintaining two univariate feature hierarchies. Furthermore, this proposed approach results in a bivariate feature representation that is orders of magnitude smaller than the raw data while still allowing full exploration of the entire parameter space.

As the bivariate graph is created in an offline preprocessing step, scientists are able to explore the parameter space for both distance and velocity similarity parameters and gain an understanding of the underlying data. Figure 4.10 provides an example of halos for the commonly used 0.2 linking length explored across velocity similarity values of 100%, 50%, and 0%. A second example for another time step of the data set is shown in Figure 4.11. As illustrated in the figures, it is apparent that as the velocity similarity decreases, particles with different velocity vectors get clustered into one halo, and thus produce inaccurate halo results. In particular, having the flexibility to define both distance and velocity similarity of particles enables accurate halo selection by separating the halos that cross paths over time. As parameter values are changed, halos are interactively extracted, and various feature-based attributes such as size and mass are calculated at run time, which provides the additional flexibility to interactively explore halos by filtering them according to their size, as shown in Figure 4.11. More results related to this particular application example can be found in [172].

4.2.2 Atmospheric Science Data - Thunderstorm Complexes 08/2011

Many weather events can be characterized by variations in surface pressure from the mean pressure value (i.e., pressure-perturbation). Accordingly, there is significant interest in extracting pressure-perturbation events to better understand the various weather events. This notion of pressure perturbation is a fairly new concept assumed to de-correlate pressure from elevation in the definition of severe weather events. The bivariate hierar-

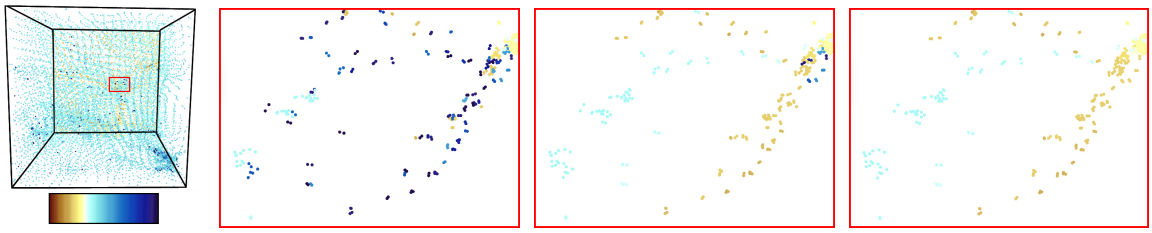


Figure 4.10: For a zoomed-in view, as indicated in red, of the halo particles in (a), halos for the commonly used 0.2 linking length are explored for varying velocity similarity values: (b) 100%, (c) 50%, and (d) 0%. Here, as the velocity similarity is decreased, the light blue and dark blue halos in the bottom left corner in (b) are clustered together as one halo. The color map used to color features is displayed alongside the results.

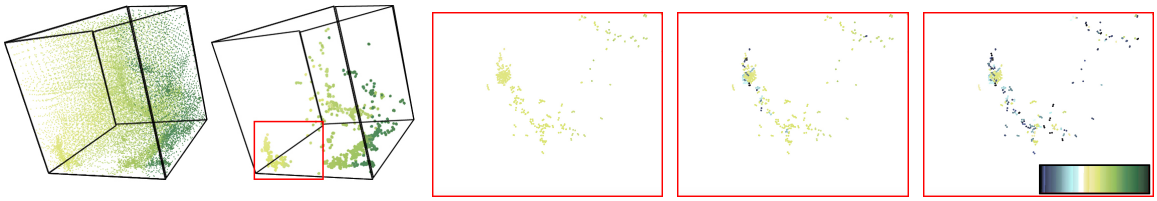


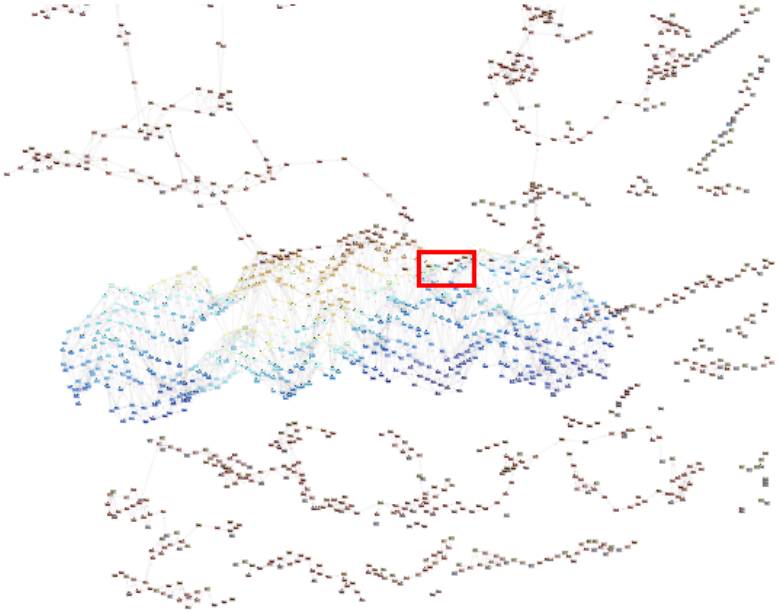
Figure 4.11: Another example where halos at 0.18 linking length are explored for different velocity similarity values: First, halos in (a) are filtered by their size ≤ 2, to result in (b). Then, halos are explored by varying the velocity similarity: (c) 0%, (d) 50%, and (e) 100%. Again, the flexibility to defining both distance and velocity similarity of particles enables more accurate halo selection. Here, the color map used to color features is displayed alongside the results.

chies are used to test this hypothesis. In particular, two hierarchies are combined, one on pressure-perturbation and one on elevation to understand if and where the two fields are correlated. Assuming the pressure perturbation is completely uncorrelated to elevation, the changes in elevation values are expected to have no effect on the final features.

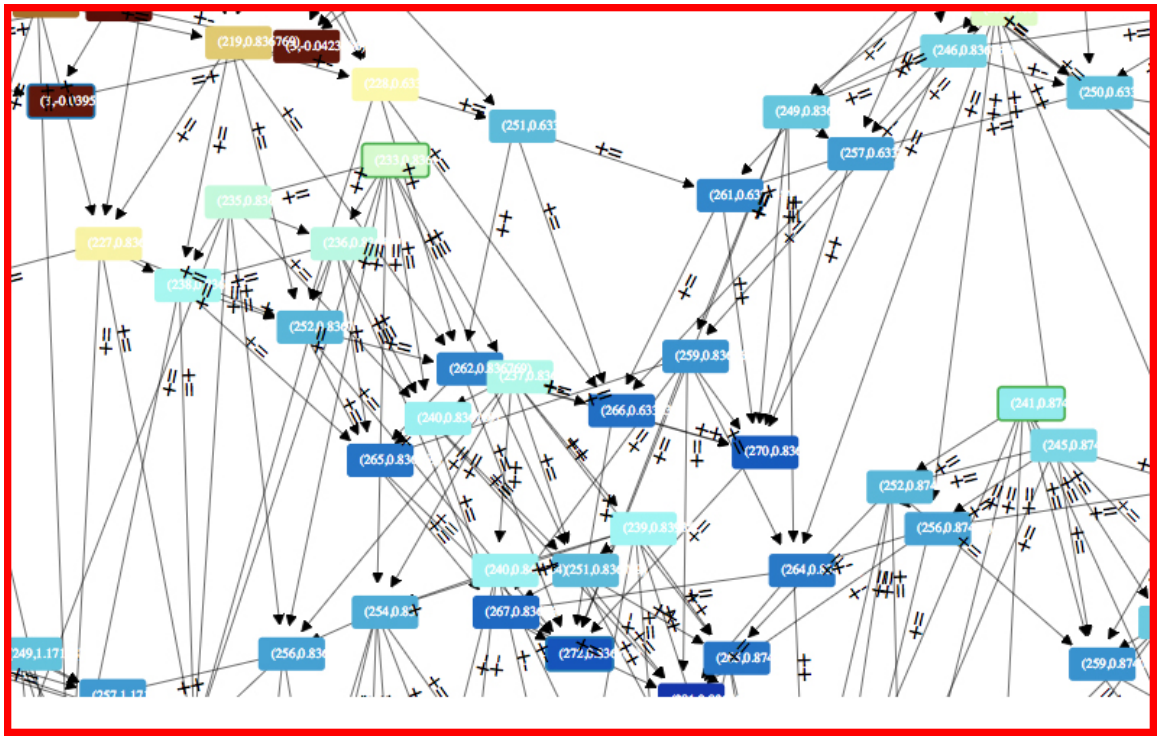
A atmospheric science data set generated by combining high-resolution grids of numerical simulations [173] with high-frequency pressure observations [174], [175] is considered. In particular, the development and movement of several distinct thunderstorm complexes over the Great Plains, which took place from 2100 Universal Coordinated Time (UTC) on August 11, 2011 through 0000 UTC on August 13, 2011, is the focus of the case study. For this data set, the pressure perturbation hierarchy for range from - 0.9 hPa to - 1.1 hPa bracketing the default parameter of - 1.0 hPa is constructed. When constructing the elevation hierarchy, all velocity values are considered.

Figure 4.12 shows a bivariate hierarchy constructed for one time step within this data set. Here, the complexity of the bivariate hierarchy indicates a lack of a meaningful correlation between pressure-perturbation and elevation.

However, since the relevant bivariate feature hierarchies are precomputed, scientists are able to interactively explore the entire feature space of both pressure-perturbation and elevation values. Such exploration allows scientists to gain insights into the two parameters. Figure 4.13 provides two examples of weather events around the - 0.25 hPa pressure-perturbation value being explored across varying elevation values. These results indicate that there is a small, although not meaningful, correlation between the two param-



(a) Bivariate hierarchy



(b) Zoomed in View

Figure 4.12: For a particular time step of the atmospheric science data set, its resulting bivariate hierarchy for pressure-perturbation and elevation parameters is shown in (a). A zoomed-in view of the hierarchy (indicated in red) is displayed in (b).

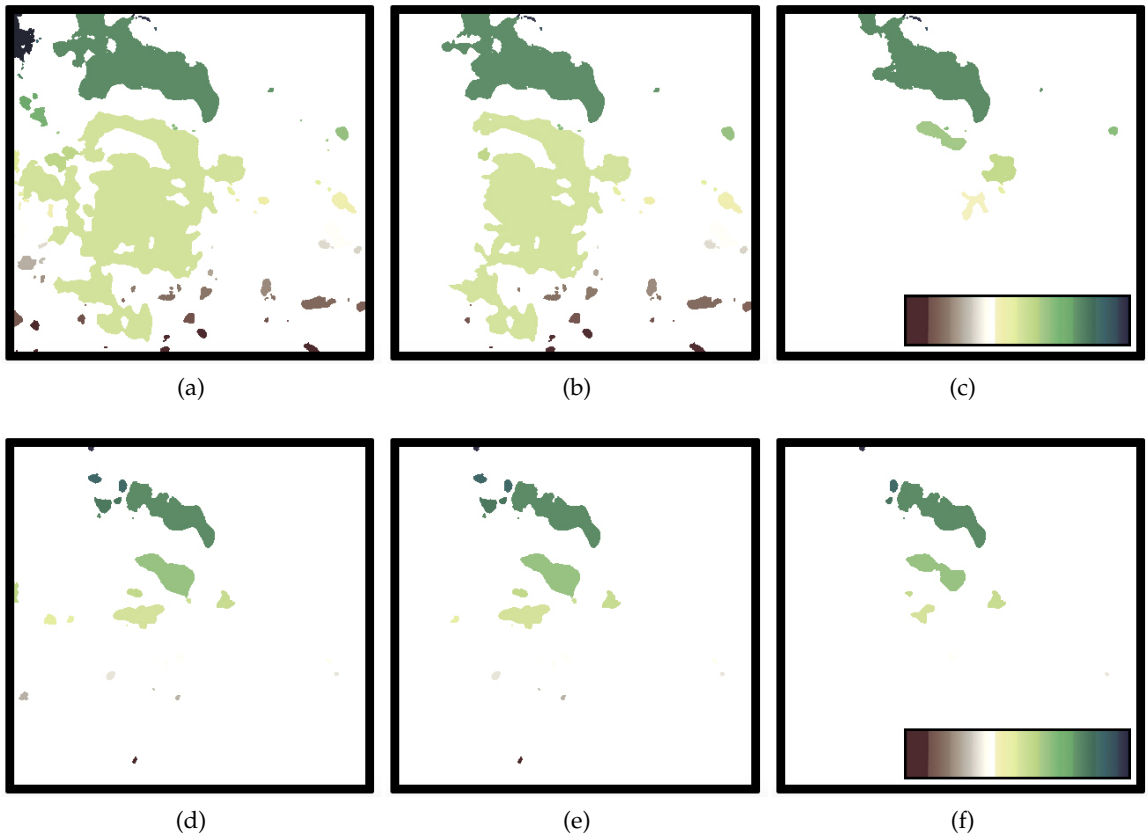


Figure 4.13: Two examples in which weather events for different pressure-perturbation and elevation values are explored. For the pressure-perturbation value - 0.25 hPa, (a) – (c) show how weather events change as the elevation value is decreased, 3500, 500, 150, respectively. Again, in (d) – (f) pressure-perturbation value is kept fixed at - 0.50 hPa, and the elevation value is varied, 3000, 1000, 250, respectively. For each example, the color map utilized to color features is displayed alongside the results.

eters. Furthermore, the results show that formulating pressure-perturbation events from pressure observations removes elevation as a major correlative factor for weather events but does not de-correlate them. More results related to this particular application example can be found in [172].

4.3 Summary

Here, a novel graph structure that efficiently and compactly encodes the hierarchical nature within bivariate features is introduced. Specifically, the hierarchical nature of the input parameters is exploited to compute the bivariate nesting of features, and the results are stored within a graph structure called bivaiate graph. Once computed, this bivariate

graph allows a direct and interactive manipulation of both input parameters, providing scientists the flexibility to explore the entire parameter space of bivariate features along with their attributes and geometric details. Here, with a focus on disjoint nested feature hierarchies, a simple yet effective approach to combine two univariate feature hierarchies into their bivariate feature representation is presented. This bivariate graph allows scientists, for the first time, to freely explore two parameter feature definitions in large-scale data sets. Finally, the usability of this bivariate graph is demonstrated with examples from cosmology and atmospheric science domains.

CHAPTER 5

COMPUTING AND ENCODING FEATURE CORRESPONDENCES AT MULTIPLE SCALES

Once features are identified, the next step toward understanding dynamic data via exploring the evolution of features is extracting feature correspondences across time. The hierarchical feature representations presented in Chapters 3 and 4 remove the need for repeated feature computation, thus making interactive feature selection possible. However, interactively extracting feature correspondences across time to compute the feature evolution details remains an expensive operation due to the costs involved with the correspondence computation step. To establish feature correspondences across time, all features in each consecutive pair of time steps in the dynamic data set have to be compared, which typically requires multiple traversals of the corresponding segmentation as well as efficient search data structures. Even for practical data sets, this usually entails processing a large number of features for hundreds of time steps. As the data sizes increase, the amount of data that need to be processed during this computation also grows exponentially, significantly increasing the costs as well. In such a setting, extracting feature correspondences within an interactive setting quickly becomes challenging.

To that end, just as with features, precomputing feature correspondences for a wide range of potential parameters and storing the results in an efficient look-up structure have the potential to allow interactive extraction of feature correspondences. This chapter addresses this problem of interactive extraction of feature correspondences across time. Specifically, this chapter introduces a new generic and flexible data representation called a *meta graph* that, similar to the hierarchical feature representations for features, encodes feature correspondences and various feature correspondence-based attributes for a range of parameter values. Here, in order to maintain its applicability across a range of applica-

tion domains, appropriate abstractions are maintained within the data representation. This chapter discusses construction, feature correspondence extraction, and other implementation details related to this data representation and presents several application results to demonstrate its utility.

5.1 Meta Graphs

Just as for features, precomputing a range of feature correspondences and encoding them in an efficient data representation ensures interactive exploration of feature evolution across time. The purpose of the meta graph is to encode, for all possible features at time T, their corresponding features in time steps $T-1$ and $T+1$. Here, due to the abstract nature of this data representation, domain experts are allowed to specify an appropriate feature correspondence metric (e.g., region overlap-based [15], [16], attribute-based [17–19]) based on the underlying data type. Maintaining this abstraction within the meta graph ensures its applicability across a wide range of application domains. Additionally, in order to compute feature correspondences, the relevant hierarchical feature representations that store the clustering hierarchy of features for each time step of the dynamic data set also need to be provided.

5.1.1 Meta Graph Construction

Once a feature correspondence metric and the relevant feature hierarchies are specified, feature correspondences can be extracted for a range of attribute values and then stored to form a meta graph. In fact, the algorithm to construct this meta graph proceeds in two steps. First, per-element correspondences are computed using the element ids stored within the feature hierarchies; second, these correspondences are converted into per-feature correspondences. Here, it is important to note that this proposed hierarchical feature representation is able to handle both feature hierarchies, *disjoint* and *overlapping*.

As discussed in Chapter 3, the feature hierarchies store element details within the hierarchy on a per-branch (for disjoint hierarchies) or per-node (for overlapping hierarchies) basis. When computing the per-element correspondences between two time steps, the specified feature correspondence metric is used to compare two elements. The simplest example is a region overlap-based metric assigning a 1 to vertices $({v_i}^T, {v_j}^{T+1})$ in consecu-

tive time steps if the regions overlap for the two vertices and a 0 otherwise. Alternatively, in the case of Twitter data, a text-based similarity metric can be used to compare tweets across time.

Once the per-element correspondences are computed, they need to be converted into per-feature correspondences. For each corresponding pair of elements $({e_i}^T, {e_j}^{T+1})$ in time steps T and $T+1$, their matching features $({f_i}^T, {f_j}^{T+1})$ are found. Then, if there does not yet exist a correspondence between ${f_i}^T$ and ${f_j}^{T+1}$, one is created with the relevant metric value (i.e., the similarity measure). If a correspondence already exists, the metric value is accumulated. For example, in the case of a region overlap-based metric, for each successful pair of overlapping vertices $({v_i}^T, {v_j}^{T+1})$ in time steps T and $T+1$, either a correspondence is made between their matching features $({f_i}^T, {f_j}^{T+1})$ with the metric value 1; or if a correspondence already exists, its metric value is increased by 1, maintaining the amount of region overlap between the two features.

These resulting feature correspondences are of two types: correspondences across features whose lifetimes overlap and correspondences across features whose lifetimes do not overlap. As discussed in Section 3.1.1.1, the lifetime of a feature within a disjoint feature hierarchy is defined by the parameter range of its corresponding branch. In the case of overlapping feature hierarchies, a feature may contain multiple branches. Therefore, a feature's lifetime is defined by the union of the parameter ranges of its branches. Conceptually, one only needs to keep correspondences between features whose lifetimes overlap, since only those features can exist simultaneously. When considering localized parameter values (refer to Figure 3.5c in Section 3.1.2), correspondences between features whose lifetimes do not overlap are needed as well. However, considering both types of feature correspondences increases the size of the meta graph immensely, and since only the former type (correspondences between features whose lifetimes overlap) is commonly used, only those correspondences are kept within the meta graph structure. When considering localized parameter values, those additional feature correspondences are computed at the tracking graph construction time.

Consequently, in order to store only the correspondences across features whose lifetimes overlap, the feature correspondences resulting from the second step of the algorithm are processed. For each corresponding pair of features ${f_i}^T$ and ${f_j}^{T+1}$ in time steps T and

T+1 with metric value $m_{({f_i}^T, {f_j}^{T+1})}$, first, it is determined whether the lifetimes of the corresponding features overlap or not. If they do not overlap, and if the feature hierarchies are non-nested, the correspondence between features ${f_i}^T$ and ${f_j}^{T+1}$ is removed. If they do not overlap, and if the input feature hierarchies are either nested disjoint feature hierarchies or nested overlapping feature hierarchies, of the two features, the feature that is lower in the feature hierarchy is picked. Then, the ancestor/ancestors of that feature, say $f_k = ancestor_{f_j}(f_i)$, whose lifetimes do overlap are found. Next, the correspondence between features ${f_i}^T$ and ${f_j}^{T+1}$ is removed and a new correspondence between the feature ${f_i}^T$ and each ancestor feature ${f_k}^{T+1}$ is added with metric value $m_{({f_i}^T, {f_j}^{T+1})}$. In this manner, once all feature correspondences are processed, only the feature correspondences across features whose lifetimes overlap are considered. Other feature correspondences are discarded and, depending on the type of feature hierarchies, additional feature correspondences across features are added.

If the input feature hierarchies are nested disjoint feature hierarchies, an additional step is performed on the resulting feature correspondences. This step considers the resulting feature correspondences (existing across features whose lifetimes overlap), and their metric values are accumulated along the hierarchy. Since features are represented by subtrees within the nested disjoint feature hierarchy, each feature that corresponds with a branch also corresponds with all its parent branches. In other words, to use the metaphor of terrain, the top of the mountain is correlated with its base. Furthermore, as discussed in Section 3.1.1, the nested feature hierarchy stores each original element along with the smallest feature that contains it (i.e., a feature contains only its exclusive elements). Therefore, the resulting feature correspondences, which are computed from the exclusive element correspondences, need to be accumulated along the hierarchy to produce the correct feature correspondences. For example, in the case of a region overlap-based metric, the region overlap of all the descendants of a certain feature is added to the correspondence of that feature to get an accurate measure. This guarantees that all ancestor features with the corresponding descendants are considered linked. For nested overlapping feature hierarchies, due to their overlapping property instead of performing this accumulation step at the meta graph construction time, it is performed at the tracking graph construction time.

Together, all these feature correspondences across the entire data set form the meta graph. Specifically, the meta graph contains nodes corresponding to all features in the data set, and each node contains edges to establish correspondences to features in adjacent time steps whose lifetimes overlap, see Figure 5.1. Additionally, just as with the univariate feature hierarchies, various feature correspondence-based attributes such as the metric value can be computed and stored within the meta graph on a per-edge basis. Likewise, given a feature correspondence metric and the relevant feature hierarchies, the meta graph structure can be used to efficiently encode the feature correspondences within a data set for a range of parameter values. More details on this meta graph construction algorithm can be found in [125].

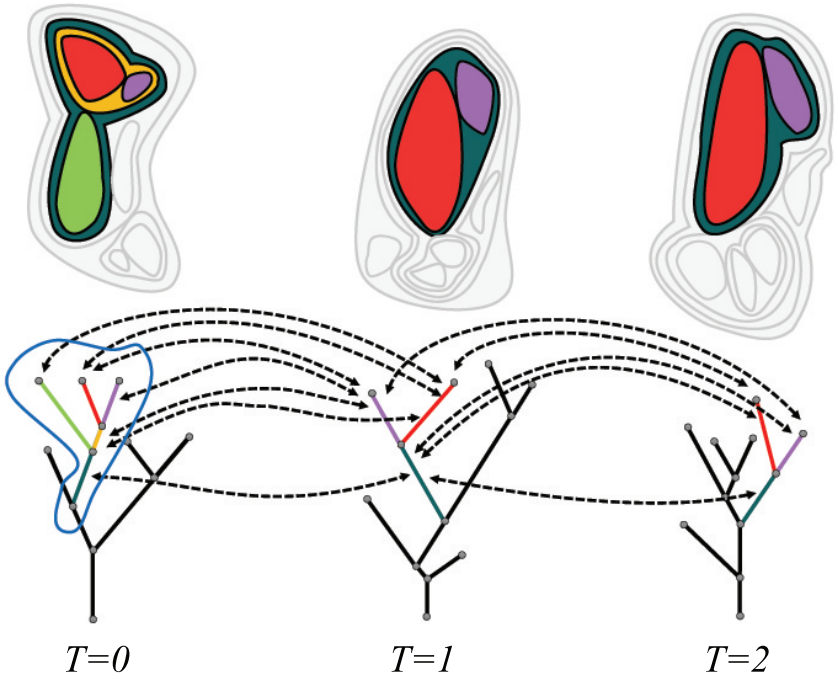


Figure 5.1: Feature correspondences stored within the meta graph structure. As indicated with arrows, for all possible features at time *T*, their correspondences to features whose lifetimes overlap in the next time step, *T+1*, are stored. For clarity, only the feature correspondences related to the highlighted branch in time $T = 0$ (outlined in blue) are displayed. Here, the lifetime of a feature is indicated by the parameter range of its corresponding branch in the disjoint feature hierarchy.

5.1.2 Feature Evolution Extraction

Given a meta graph and its corresponding feature hierarchies, feature evolution details can be quickly and easily extracted for a particular parameter setting. For a parameter value f_1, first the feature hierarchies are queried to extract the features at this parameter value, and then all edges across those features are extracted from the meta graph structure. Together, these extracted features and edges form the feature evolution details at f_1, and can be easily visualized using a tracking graph, see Figure 5.2.

For all feature hierarchies except nested overlapping feature hierarchies, these edges extracted from the meta graph represent the correspondences across features at parameter value f_1. As previously mentioned, the accumulation step within the meta graph construction algorithm is skipped for nested overlapping feature hierarchies. Therefore, in order

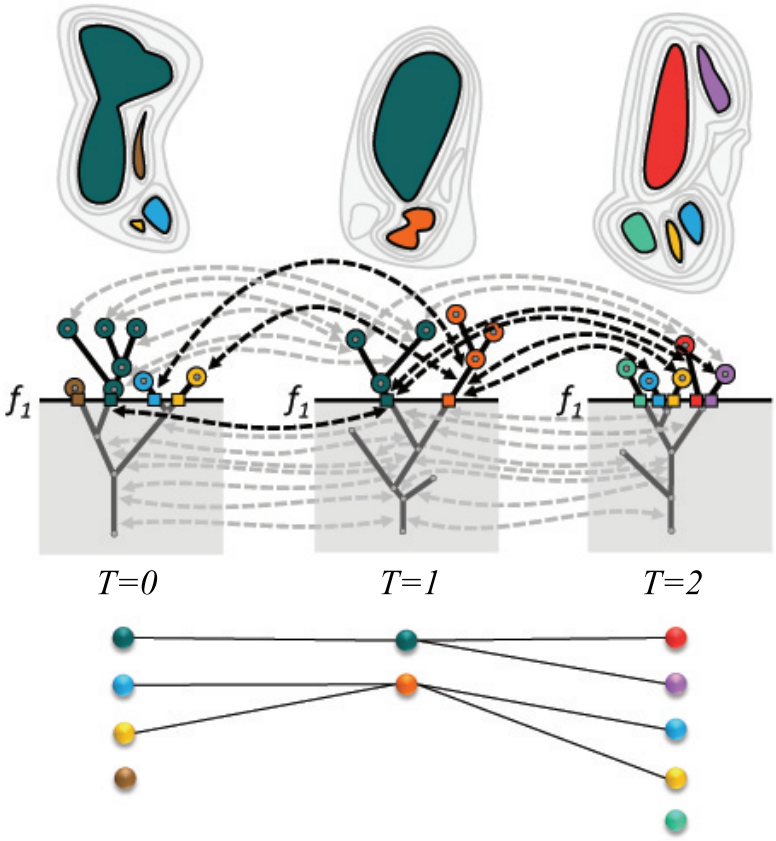


Figure 5.2: A tracking graph for a fixed parameter value f_1 is obtained by first extracting the features at f_1 from the disjoint feature hierarchies and then their correspondence details from the meta graph. Here, the extracted feature correspondences are indicated by black arrows.

to obtain the accurate feature correspondences, at the tracking graph construction time, all edges across the children of those features should also be extracted in addition to the edges extracted earlier, which are all the edges across the features at f_1. Then, all these extracted edges need to be accumulated on a per-connected component basis.

Furthermore, the meta graph structure is flexible enough to analyze feature evolution details for localized parameter values. Consider the example in Figure 5.3. Here, the parameter value at time $T = 1$ is lowered to f_2, whereas the parameter values in the two consecutive time steps are kept at f_1. Just as earlier, when obtaining the feature evolution

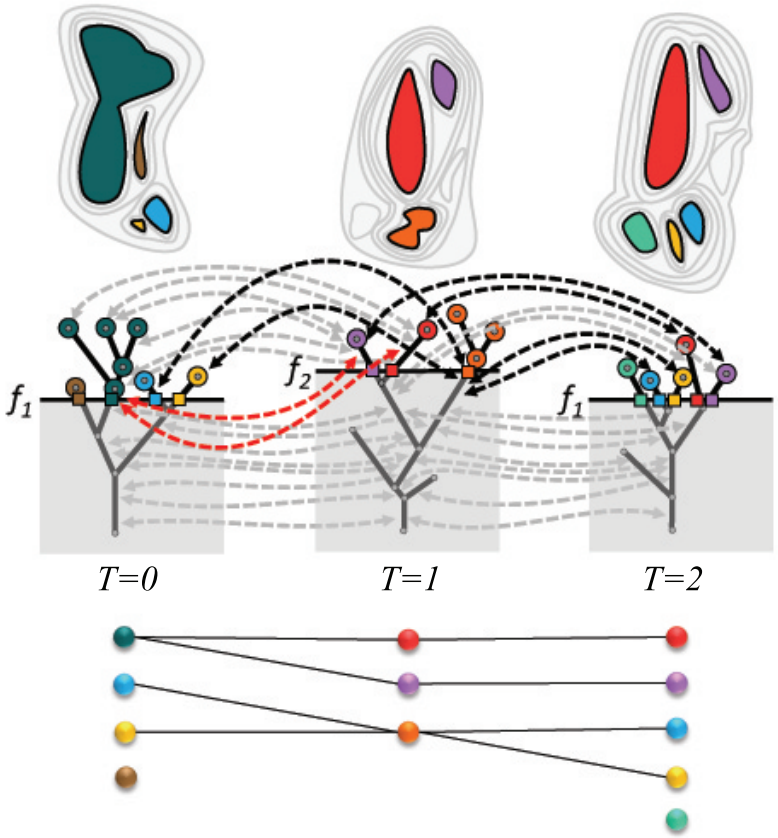


Figure 5.3: Extraction of a tracking graph with localized parameter values. For the same example in Figure 5.2, the parameter value at time $T = 1$ is lowered to f_2. When obtaining the tracking graph, features at parameter value f_2 in time $T = 1$ are extracted, and their correspondences with features at parameter value f_1 in times $T = 0$ and $T = 2$ are obtained. Here, the feature correspondences not stored within the meta graph (i.e., correspondences across features whose lifetimes do not overlap) are computed at the tracking graph construction time (indicated by the red arrows), and the other correspondences are obtained from the meta graph (indicated by the black arrows).

details, first features at each time step are extracted using the feature hierarchies. Then, all edges across those features are extracted from the meta graph. As some features at f_1 in time $T = 0$ and at f_2 in time $T = 1$ do not have overlapping lifetimes, their correspondences are not stored within the meta graph. Hence, those feature correspondences (indicated by the red arrows in Figure 5.3) are computed at the tracking graph construction time. Specifically, the same aforementioned construction approach in [125] is followed, and feature correspondence details are computed. Here, if the feature hierarchies are nested, by making use of the existing feature correspondences in the meta graph, edges for only a subset of the feature hierarchy have to be computed. Then, those edges are accumulated with the existing edges. The same process is performed across features at f_2 in time $T = 1$ and f_1 in time $T = 2$. Together, these edges that are computed at the tracking graph construction time and the earlier extracted features and edges form the final tracking graph.

Another example is illustrated in Figure 5.4. Here, the parameter value for a selected feature at time $T = 1$ is lowered to f_2 whereas the parameter values at other features are kept at f_1. Again, the feature correspondences not stored within the meta graph (indicated by the red arrows in Figure 5.4) are computed at the tracking graph construction time. For all these cases, in addition to the feature evolution details, feature correspondence-based attributes stored within the meta graph can also be extracted interactively.

5.1.3 Implementation Details

The meta graph encodes all possible feature correspondences across features whose lifetimes overlap in adjacent time steps of the entire dynamic data set. Once a feature correspondence metric and the relevant feature hierarchies are specified for a given data set, an offline preprocessing step is used to compute this meta graph. As mentioned earlier, various feature correspondence-based attributes are computed and stored along the meta graph on a per-edge-basis at the construction time. Once constructed, the meta graph details are stored in multiple files (i.e., one file per time step), each containing a set of edges representing its feature correspondences with the next time step.

When exploring feature evolution details, individual files are read as necessary and stored within a map data structure (i.e., one map data structure per time step). For a

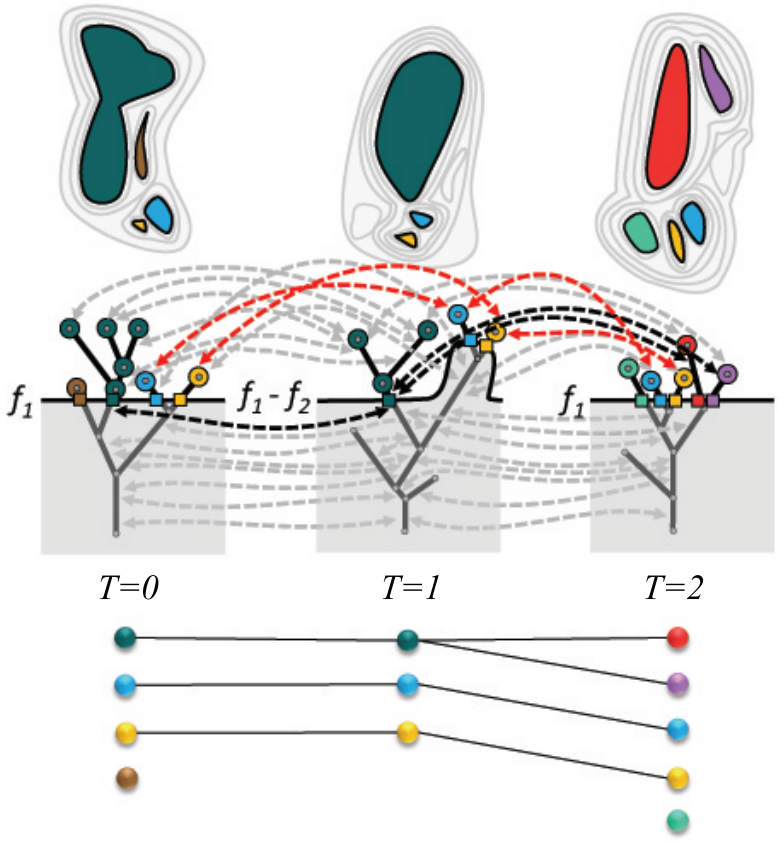


Figure 5.4: Effects of using localized parameter values on selected features. For the same example in Figure 5.2, the parameter value for the high valence, merge-split node in time $T = 1$, is lowered until it is split into individual nodes. When computing the tracking graph, feature correspondences across those individual nodes in time $T = 1$ and features in times $T = 0$ and $T = 2$ are obtained. Again, the feature correspondences not stored within the meta graph are computed at the tracking graph construction time (indicated by red arrows), and the rest are extracted from the meta graph (indicated by black arrows).

particular parameter setting, first, for each time step in the data set, all living features are extracted from their feature hierarchies. Then, for each extracted feature within a time step, its correspondences with the other extracted features in the next time step are obtained from the meta graph. Since the necessary feature correspondences are precomputed and can be accessed through efficient look-ups, all feature evolution details can be extracted on the fly. When localized parameter values are used, feature correspondences not stored within the meta graph (i.e., correspondences across features in adjacent time steps whose lifetimes do not overlap) are computed at the tracking graph construction time. Additionally, as feature correspondence-based attributes are stored within the meta graph on a

per-edge basis, meta graphs provide the ability to filter feature evolution details according to various feature correspondence-based attributes, which effectively changes the feature corresponding criterion on the fly.

5.1.4 Advantages and Limitations

The meta graph is a generic data representation that can be applied across a range of application domains. Once an appropriate feature correspondence metric is specified for a dynamic data set, it is able to encode feature correspondences for a range of parameter values. Along with the feature correspondences, various feature correspondence-based attributes can also be stored within this data representation. Consequently, when the corresponding feature hierarchies are available, the meta graph has the capability to interactively explore feature evolution across time. Here, it is important to note that, given both types of feature hierarchies, disjoint and overlapping, the meta graph has the capability to store feature correspondences and explore the feature evolution across time. However, for nested overlapping feature hierarchies, additional computation at the tracking graph construction time is needed to obtain the accurate feature correspondences.

This data representation is also flexible enough to ensure interactive extraction of feature correspondences for both fixed and adaptive parameter values. As the necessary feature correspondences are precomputed and stored within the meta graph, feature evolution details can be extracted for fixed parameter values without any additional computation. However, for adaptive parameter values, additional computation is needed to find correspondences across features whose lifetimes do not overlap. This additional computation needed can be considered as a drawback of the meta graph structure, but if all possible feature correspondences are stored within the meta graph, its size increases immensely. Instead, storing the feature correspondences across only features whose lifetimes overlap and computing the other correspondences as necessary at the time the tracking graph is constructed result in amenable data sizes and ensures interactivity. By making use of the existing feature correspondences in the meta graph, edges for only a subset of the feature hierarchy have to be computed at this time.

5.2 Application Results

In this section, application results from atmospheric science and plasma surface interaction domains are presented to demonstrate the effectiveness of the meta graph structure.

5.2.1 Atmospheric Science Data - Thunderstorm Complexes 08/2011

Understanding how various weather systems develop and evolve across time is beneficial to atmospheric scientists from both research and operational (e.g., weather forecasting) perspectives. Several atmospheric state variables can be measured to identify high-impact weather events, one of which is surface atmospheric pressure. Many weather systems are characterized by variations in surface pressure from the mean pressure value (i.e., pressure-perturbations). Accordingly, there is significant interest in extracting and tracking pressure-perturbation occurrences both spatially and temporally to better understand the evolution of weather events.

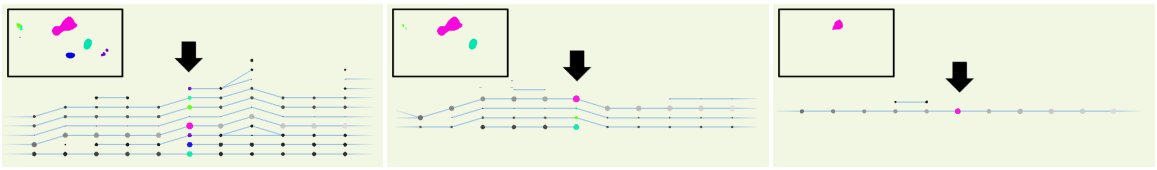
Here, the same case study considered in Section 4.2.2 is considered. This particular case study is extracted from a pressure-perturbation data set generated by combining high-resolution grids of numerical simulations [173] with high-frequency pressure observations [174], [175]. It involves the development and movement of several distinct thunderstorm complexes over the Great Plains, and took place from 2100 Universal Coordinated Time (UTC) on August 11, 2011 through 0000 UTC on August 13, 2011. The original data set for this case study consists of 468 time steps and totals about 850MB.

For this data set, the relevant feature hierarchies and meta graph structures are computed in an offline, preprocessing step. Once the feature hierarchies are computed to store the clustering hierarchy of pressure-perturbation events for a range of pressure-perturbation values, the total data size is reduced to $\approx 60MB$. A threshold-based segmentation is used to compute these feature hierarchies. During construction, various feature-based attributes including centroid, area, median magnitude value, maximum/minimum magnitude value, position of the maximum/minimum magnitude value, long-axis distance, short-axis distance, long-axis orientation, eccentricity, propagation distance, and propagation speed are also computed and stored within the feature hierarchies. It is important to note that a vast majority of these data is used for storing the list of feature-based

attributes required for feature extraction and spatial information needed for rendering. For this case study, the collaborating scientists are particularly interested in exploring the evolution pressure-perturbation events with respect to both region overlap and distance proximity-based metrics. Therefore, to support their needs, feature correspondence details for each metric are computed and stored within separate meta graph structures. The feature correspondence amount (i.e., the amount of region overlap or the length of Euclidean distance between features) is stored as a feature correspondence-based attribute within each meta graph. At the end of this step, the meta graph files total around 10MB.

Since the relevant nested feature hierarchies and meta graph structures are precomputed, atmospheric scientists are able to interactively explore the entire feature space by varying the pressure-perturbation value. Such exploration allows scientists to gain insights into which pressure-perturbation parameter range is reasonable for exploring a particular pressure-perturbation event. Figure 5.5 provides an example in which pressure-perturbation events near the commonly used 1.0 hPa pressure-perturbation value are explored. Several pressure-perturbation events in the spatial domain and a portion of their corresponding tracking graphs across pressure-perturbation values of 0.75 hPa, 1.0 hPa, and 1.25 hPa are displayed in this example. Here, it is apparent that as the pressure-perturbation value increases, pressure-perturbation events decrease in both number and size (with only stable pressure-perturbation events remaining), and the resulting tracking graph reduces its complexity.

The use of tracking graphs allows scientists to gain a global, concise view of the case study. These graphs provide insights into the underlying structure of a particular pressure-



(a) 0.75 hPa (b) 1.0 hPa (c) 1.25 hPa

Figure 5.5: Effects of varying the pressure-perturbation value to explore the entire feature space. The pressure-perturbation events and a portion of their corresponding tracking graphs are shown at (a) 0.75 hPa, (b) 1.0 hPa, and (c) 1.25 hPa pressure-perturbation values. In each case, the focus time step of the tracking graph is indicated with a black arrow.

perturbation event (e.g., its complexity, duration, and other trends). As shown in Figure 5.6, the resulting graphs indicate that several different weather systems exist in this case study and that they evolve (for the most part) distinctly over time.

Furthermore, the flexibility in the meta graph to extract feature evolution details for localized parameter values is of much value. It has the potential to allow atmospheric scientists to adaptively change the pressure-perturbation value over time to produce better, more consistent tracking results. For this case study, scientists are particularly interested in analyzing the evolution of the feature with a bow-like structure (the selected feature in Figure 5.6). A portion of the global tracking graph for this case study is shown in Figure 5.7a, and the feature with a bow-like structure is indicated in yellow. According to the tracking graph in Figure 5.7a, this feature exists for a long period of time, from $T = 100$ to $T = 145$ to $T = 165$, and then disappears suddenly at $T = 165$ (i.e., $T = 100$ to $T = 165$ is 325min or 5.4hr). It then appears again at $T = 170$ and evolves from that point on. Further investigation reveals that this sudden disappearance and reappearance of the feature is due to variations in the pressure-perturbation value between $T = 165$ and $T = 170$ and the chosen pressure-perturbation value.

Here, slightly reducing the pressure-perturbation value for all time steps $166 \leq T \leq 169$ (e.g., 0.88 hPa at $T = 168$) allows scientists to obtain an easily comprehensible tracking graph, where this bow-like feature appears consistently across consecutive time steps between $165 \leq T \leq 170$, see Figure 5.7b. In particular, for each particular time step $T \in (165, 170)$, an arbitrary cut within the nested feature hierarchy is obtained. The arbitrary cut obtained for $T = 168$ is indicated by the black vertical curve in Figure 5.8. This flexibility in defining localized, per-feature parameter values can provide scientists insights into features of interest, in this case the bow-like structure. Specifically, for a small

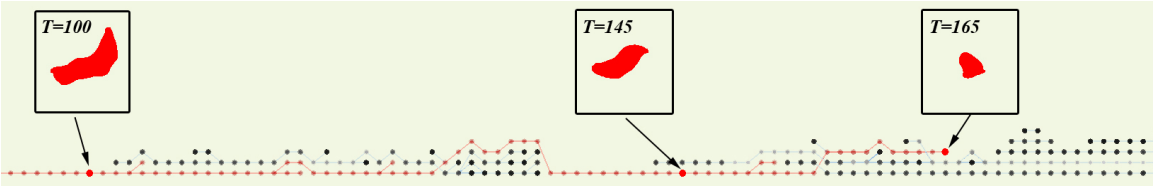
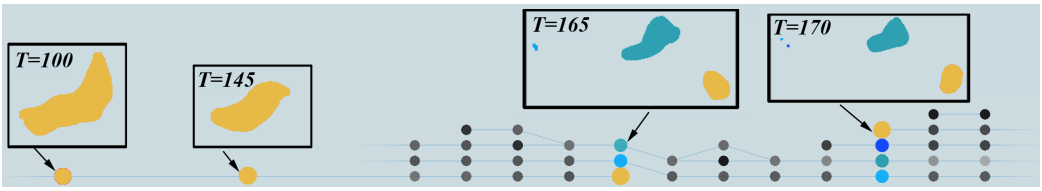
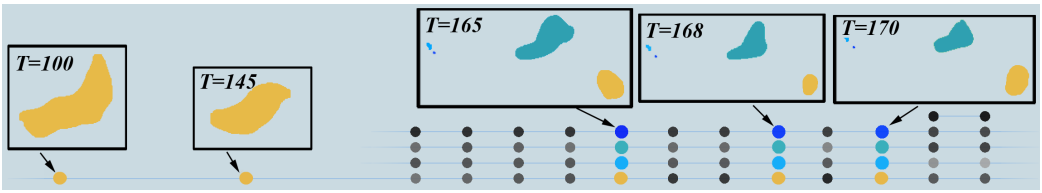


Figure 5.6: A longer tracking graph showing the evolution of pressure-perturbation events at 1.0 hPa. The evolution of a feature with a bow-like structure is highlighted in red.



(a) Tracking graph at 1.0 hPa



(b) Same tracking graph using adaptive parameter values

Figure 5.7: An example showing the use of localized parameter values to extract flexible feature evolution details. (a) A tracking graph at 1.0 hPa showing the sudden disappearance and reappearance of a feature with a bow-like structure (in yellow). This feature disappears at $T = 165$ and then reappears at $T = 170$. Exploring the feature hierarchies reveals that this is likely due to variations in pressure-perturbation between time steps. (b) Therefore, by locally modifying the feature parameter values for those time steps (from $T = 165$ to $T = 170$), a much simpler tracking graph, in which this feature is stably evolving, can be obtained. In both cases, features at several time steps are visualized.

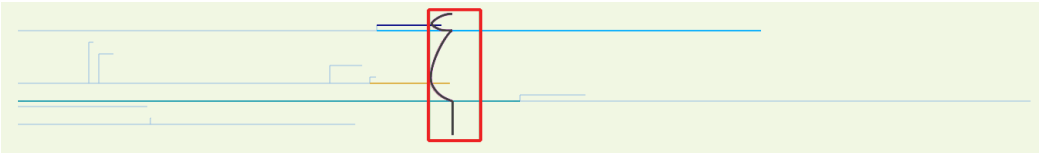


Figure 5.8: For the same tracking graph as in Figure 5.7b, the horizontal graph layout of the feature hierarchy at $T = 168$. For this particular time step, the feature hierarchy is a forest with a pressure-perturbation range 0.9 – 1.1 hPa (increasing from left to right). The localized, per-feature parameter values are obtained by creating an arbitrary cut (indicated by the black vertical curve) within a user-defined parameter range (indicated by the red square).

pressure-perturbation value range around 1.0 hPa, this particular feature exists continuously for a longer period of time than at a fixed pressure-perturbation value (i.e., 1.0 hPa). More results related to this particular application example can be found in [176].

5.2.2 Plasma-Surface Interactions Data

Here, a data set collected from an atomistic simulation of plasma-surface interactions is considered. These simulations have shown that helium spontaneously aggregates to form clusters and eventually bubbles, pushing out tungsten surface defects (i.e., voids

or cavities) in the process [177–179]. Exploring the evolution of these helium bubbles to understand the origin of fuzz-like, microscopic damage to tungsten and other metal surfaces by helium has recently been the focus of significant research. As such, the helium bubbles are considered to be the features of interest for this type of data. Furthermore, the clustering hierarchy of helium bubbles is computed using a distance-based clustering algorithm.

For this particular case, instead of exploring the helium bubble evolution for a range of distance values, the collaborating scientists are interested in exploring the evolution details for a specific distance value, $\sqrt{3}a/2 \approx 2.75$ Å. Therefore, the nested feature hierarchies are created for that fixed distance value. In particular, the tightly coupled simulation–visualization pipeline proposed by Widanagamaachchi et al. [180] is used to compute the required nested feature hierarchies at distance value ≈ 2.75 Å. The resulting data set contains 326 time steps totaling about 18GB of data. Once the feature hierarchies are obtained, the meta graph structure is computed using a region overlap-based feature correspondence metric. The resulting data files storing the meta graph structure total about 12MB. Together, the meta graph and feature hierarchies enable interactive exploration of helium bubble evolution across time.

One of the processes observed in this data set is a rupture of an over-pressurized helium gas bubble that is near the tungsten surface. As shown in Figure 5.9, this process is easily observed through visualization. However, due to the complex interactions, the evolution of the helium bubbles around this bubble burst is not easily perceived. Using the data stored within the meta graph structure, the evolution of those helium bubbles can be explored via tracking graphs, as shown in Figure 5.10. More results related to this particular application example can be found in [180].

5.3 Summary

This chapter discusses how interactive extraction of feature correspondences can be achieved for features via a novel data representation named meta graph. Specifically, the meta graph encodes all possible feature correspondences across features whose lifetimes overlap in adjacent time steps of a dynamic data set. In addition to the feature correspondences, various feature correspondence-based attributes are also stored within this nested

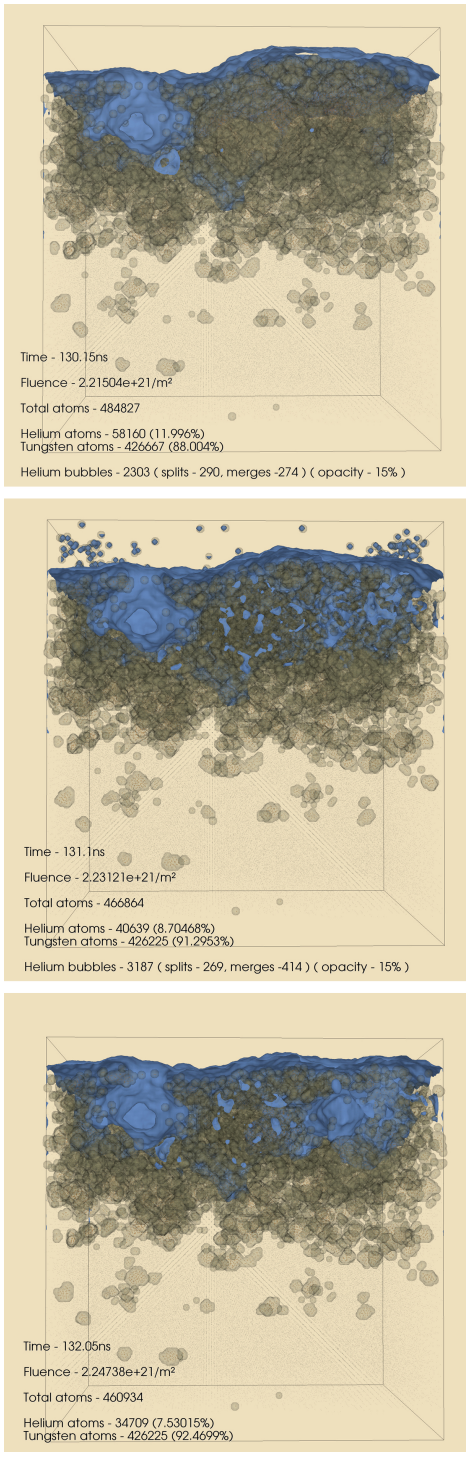


Figure 5.9: Snapshots around the time the tungsten surface is pushed upward by an over-pressurized helium bubble (top), and then recoils as the bubble bursts (center, bottom). In each snapshot, the tungsten voids are indicated in blue, helium bubbles in gray, and the extracted statistics and bubble evaluation details are also displayed. Here, the simulation box is $\approx 20\ nm$ wide.

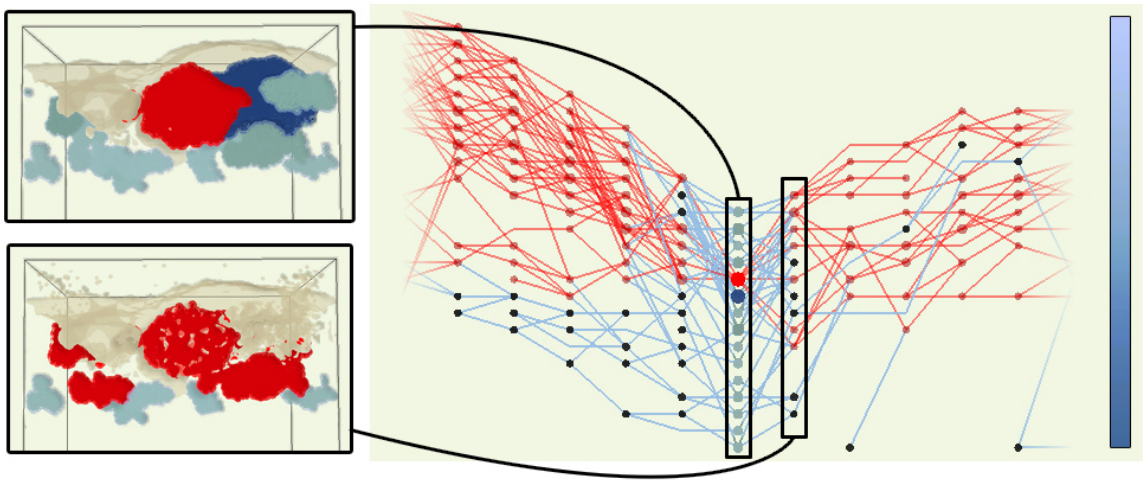


Figure 5.10: A tracking graph (right) visualizing the evolution of helium bubbles around the time one helium bubble bursts, the same as in Figure 5.9. The top left and bottom left images visualize the helium bubbles (color-coded by size using a blue color map) and tungsten voids (in beige) for two time steps. The bubble that bursts and its evolution are highlighted in red. For simplicity, only bubbles with size$\geq$500 atoms are visualized.

feature representation. As a result, once the meta graph and its corresponding nested feature hierarchies are provided, they have the capability to interactively explore feature evolution over time for a range of parameter values in a general and flexible manner. The abstract nature of the meta graph enables it to be used across a wide variety of application domains. This chapter presents the construction, feature evolution extraction, and other implementation details related to the meta graph structure and also discusses its advantages and limitations. Finally, several application examples from atmospheric science and plasma-surface interactions domains are presented to demonstrate the utility of this data representation.

PART III

INTERACTIVE EXPLORATION OF FEATURE EVOLUTION

CHAPTER 6

LAYOUT AND VISUALIZATION OF FEATURE EVOLUTION

To better understand the dynamic nature of data, it is important to make use of visual representations that can present feature evolution details within a dynamic data set in a comprehensible manner. Among the many visual representations in the literature for the purpose of visualizing feature evolution, this dissertation focuses on exploring feature evolution across time with the aid of tracking graphs. As discussed in Chapter 1, tracking graphs capture concise representations of feature evolution as a collection of feature tracks, see Figure 1.2. These graphs use constrained graph layouts with one spatial dimension to indicate time and show the tracks of each feature as it evolves, merges, and/or disappears. Due to their timeline-based approach, tracking graphs have the capability to present global overviews of feature evolution for the entire data set in a simple and effective manner. Additionally, as illustrated in Figure 5.2 in Section 5.1.2, the features and edges extracted from the meta graph and its corresponding feature hierarchies can be easily visualized with a tracking graph for a specific parameter value. Therefore, use of tracking graphs to visualize feature evolution across time fits well with the flexible data representations discussed in this dissertation.

Many data sets involve thousands of features for hundreds of time steps, and therefore, tracking graphs can quickly become incomprehensibly large and complex to comprehend. The sheer number of nodes and edges existing within the tracking graph makes the graph drawing very challenging. In particular, computing optimal or near-optimal layouts for a large tracking graph can be expensive, involving large amounts of memory and time requirements, which can severely hinder the use of tracking graphs within an interactive setting. Therefore, alternative strategies that can improve the comprehensibility and interactivity of tracking graphs, especially for larger data sets, are of utmost importance.

In order to handle large tracking graphs, this chapter focuses on progressive techniques to both lay out and visualize tracking graphs. Specifically, tracking graphs are always processed with respect to a focus time step and a window of interest. Furthermore, a progressive two-stage layout algorithm is introduced for computing interactive graph layouts for tracking graphs. This chapter discusses the particulars of this progressive graph layout and visualization strategy and presents application results to demonstrate its utility.

6.1 Progressive Graph Layout and Visualization

When exploring feature evolution within a dynamic data set, one rarely needs to look at all features across all time steps simultaneously. Accordingly, processing tracking graphs with respect to a focus time step and a window of interest is more appropriate. Starting from a time step of interest, tracking graphs can be progressively visualized both forward and backward in time up to a specific time window. Such a use of a focus time step and a window of interest limits the focus of interest to a certain subregion of the global tracking graph, enabling one to interactively change time steps, expand and contract the window of interest, and thus explore the entire tracking graph.

However, a progressive visualization strategy alone is not enough to maintain the interactivity when exploring larger tracking graphs. Interactive graph layout algorithms should also be maintained. To that end, this chapter introduces a two-stage graph layout algorithm that employs a fast initial layout and a slower greedy one on tracking graphs. As a result, a tracking graph is immediately visualized using an initial graph layout, and this layout is replaced by a better one, with fewer edge intersections, as soon as it is available.

6.1.1 Initial Graph Layout

The first stage in the two-stage graph layout algorithm is visualizing the tracking graph using an initial graph layout. As discussed in Chapter 3, the feature hierarchies that store the clustering hierarchy of features can be represented by a tree or a graph. The layout for the feature hierarchy is most commonly computed by a hierarchical graph layout that follows a depth-first ordering of its features. This hierarchical graph layout usually places the root in the middle and its subtrees recursively on either side. This fact can be exploited to quickly create a reasonable initial graph layout for tracking graphs.

Consider the example in Figure 6.1. Here, the horizontal graph layout of the feature hierarchy is considered. For each time step within a tracking graph, all its features can be placed by looking at the position of the features in the horizontal graph layout of the feature hierarchy. This choice appears to be random and not clearly optimal, but the resulting tracking graph layout is better than expected. The reason is that all feature hierarchies come from a continuously evolving simulation, and are constructed by the same code that processes the data in the same order. Therefore, these hierarchies naturally retain some temporal coherence, making the depth-first layout a good initial choice.

The initial layout for an overlapping feature hierarchy can be constructed in a similar

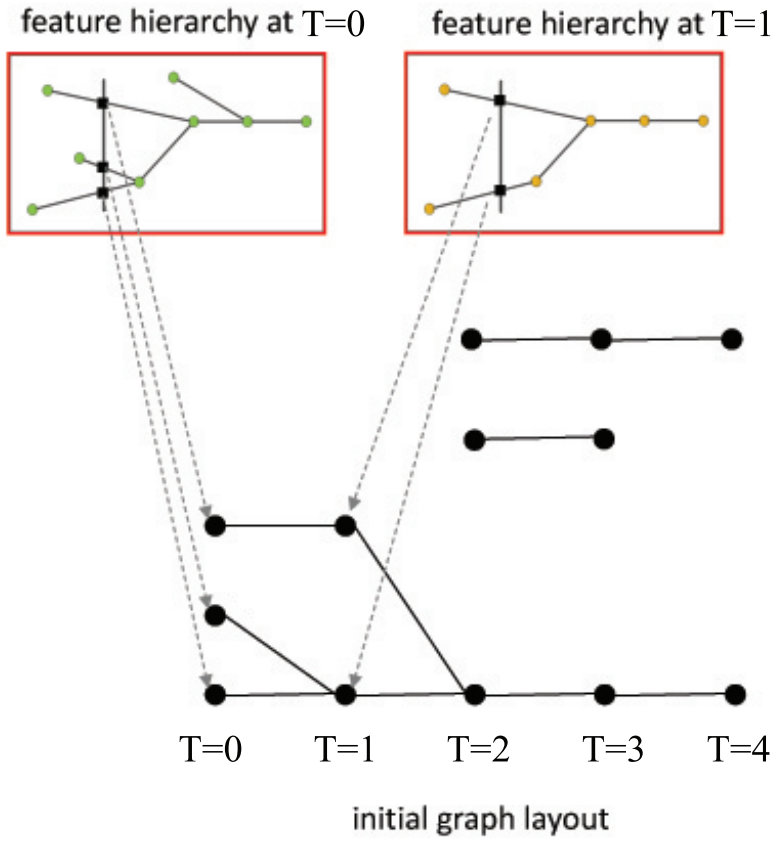


Figure 6.1: The initial layout produced during the two-stage graph layout algorithm. For each time step within the tracking graph, its features are placed by looking at the position of those features in the hierarchical graph layout of the feature hierarchy for that time step. Specifically, the horizontal layout of the hierarchy is considered here. In practice, the order of the positions of these features is considered, and then the spacing among the features is adjusted to be equal.

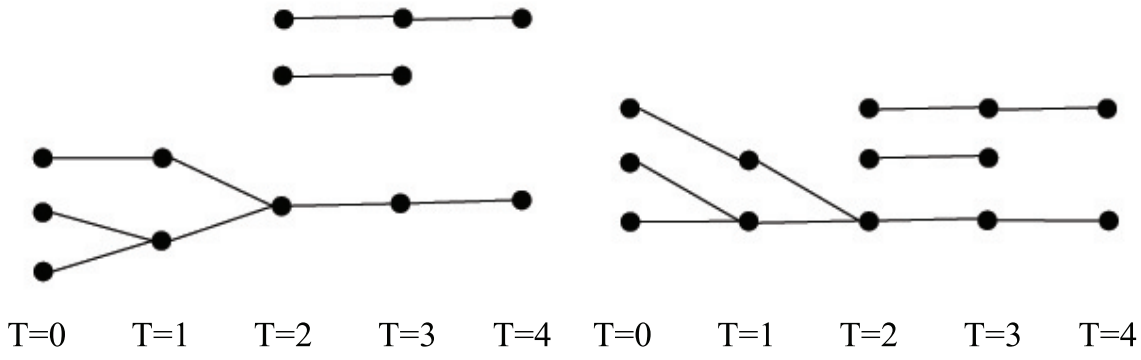
manner. As discussed in Chapters 3 and 4, overlapping feature hierarchies result in a graph instead of a tree, and their layout can be computed using the graph's hierarchical graph layout. Then, just as earlier, for each time step within a tracking graph, all its features can be placed by looking at the position of those features in the hierarchical graph layout of the feature hierarchy for that time step.

6.1.2 Greedy Graph Layout

While the time steps of the tracking graph are progressively laid out and visualized using the initial graph layout, a greedy layout with fewer edge intersections can be computed in the background. Here, although it is possible to use global layout techniques on all currently visualized time steps, the result would be in the order $O(T^2)$ as the first three, then the first five, first seven, etc. time steps are considered. Instead, a layered optimization scheme that moves outward from the focus time step is used to compute this greedy graph layout. In particular, the layout of the 'inward' time step is assumed to be fixed, and the median heuristic [112] is used to place the new nodes. The median heuristic places each node in the next time step at the median of all nodes to which it is connected in the current time step.

This optimization is aimed at minimizing edge length, and thus reduces edge crossings within the tracking graph. However, heuristics such as this are known to work well for shorter sequences, which typically are far less constrained than in the global tracking graph. As such, for smaller graphs, it is quite common for the greedy graph layout to be entirely free of intersections.

Another important aspect that is considered for the greedy layout is the placement of newly created nodes not connected to the current time step. One method used, for example by the popular Dot system [35], is to place such nodes on the top (or bottom) of the graphs. However, such a node placement strategy results in graphs with extreme aspect ratios. Therefore, with this proposed greedy graph layout, a list of 'empty' positions left by the median heuristic is maintained for placing new nodes, and thus results in more compact and visually appealing graph layouts. In practice, at the end of the algorithm, for each time step in the tracking graph, the order of the feature positions is considered, and their spacing is adjusted to be equal, see Figure 6.2.



(a) Greedy layout (b) With equal spacing

Figure 6.2: For the same graph as in Figure 6.1, the greedy layout produced during the two-stage graph layout algorithm. (a) Starting from the focus time step, $T = 0$, the median heuristic is used to place each node in the next time step at the median of all nodes to which it is connected in the current time step. (b) Then, just as earlier, the order of these feature positions is considered, and their spacing is adjusted to be equal for each time step.

6.1.3 Advantages and Limitations

As the data sizes increase, it becomes challenging to both lay out and visualize tracking graphs interactively. Here, use of a focus time step and a window of interest is beneficial as it allows one to focus on subregions of the global tracking graph. It also enables interactive exploration of the entire tracking graph by changing the focus time step, and by expanding and contracting the window of interest.

In addition to the focus time step and the window of interest, a two-stage layout algorithm is used to compute graph layouts for tracking graphs. One main advantage of this two-stage layout algorithm is its capability to produce progressive results. The initial layout of the two-stage layout algorithm is computed based on the horizontal layout of the feature hierarchy. It is important to note that, since the structure of the feature hierarchy is fixed for each time step, its horizontal graph layout can be computed once in an offline preprocessing step. Therefore, for every parameter change, the initial layout of the tracking graph can be quickly and easily computed without any computation. Specifically, the greedy layout uses a layered graph drawing algorithm coupled with a median heuristic. This layered scheme fits well with the other progressive visualization techniques used on tracking graphs, and thus allows results to be progressively integrated with the final

tracking graph. Moreover, the median heuristic works well for shorter sequences, which typically are far less constrained than the global tracking graph. However, as the time steps increase, the median heuristic pushes the edge intersections outward, without resolving them, and thus the resulting graph layout is not optimal.

6.2 Application Results

In this section, results from two combustion data sets are presented to demonstrate the effectiveness of the progressive graph layout and visualization strategy discussed in this chapter.

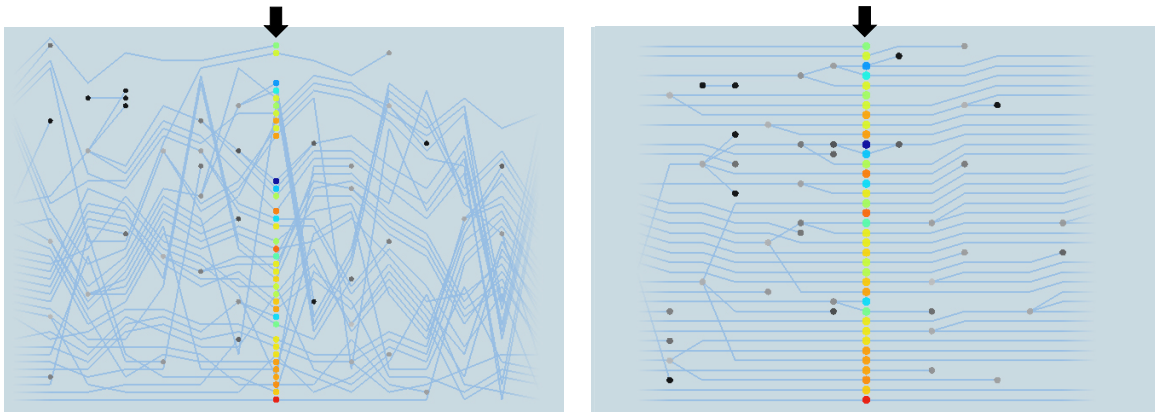
6.2.1 Combustion Data - Hydrogen Flame with No Turbulence

First, the same hydrogen flame combustion data set in Section 3.2.2 is considered. The features of interest for this data set are the burning cells. The clustering hierarchy of these burning cells is computed using threshold-based segmentation. For this data set, the meta graph structure is computed using a region overlap-based feature correspondence metric.

Figure 6.3a shows a small portion of a tracking graph in both the initial as well as the greedy layouts. Clearly, the initial layout is inferior, creating a large number of edge crossings, yet it still proves valuable for large tracking graphs as it provides immediate feedback about the size and complexity of the tracking graph. In addition, even in the initial layout there exist noticeable subgroups of nodes that behave similarly, and are laid out in parallel as a result of the natural temporal coherency present in the data.

The median heuristic used within the greedy layout is known to work well for shorter sequences, which typically are far less constrained than in a global tracking graph. Consequently, as illustrated in Figure 6.3b, the greedy layout can return layouts of smaller graphs entirely free of intersections. Even for larger graphs, the greedy layout is able to minimize edge crossings. In fact, the layered scheme used by the greedy layout pushes the edge intersections within the tracking graph outward of the focus time step, without resolving them.

For another simpler tracking graph from this data set, Figure 6.4 compares a greedy layout produced by the two-stage graph layout algorithm with the one produced by Dot. For this particular example, the two graph layouts are almost the same, with the only



(a) Initial graph layout (b) Greedy graph layout

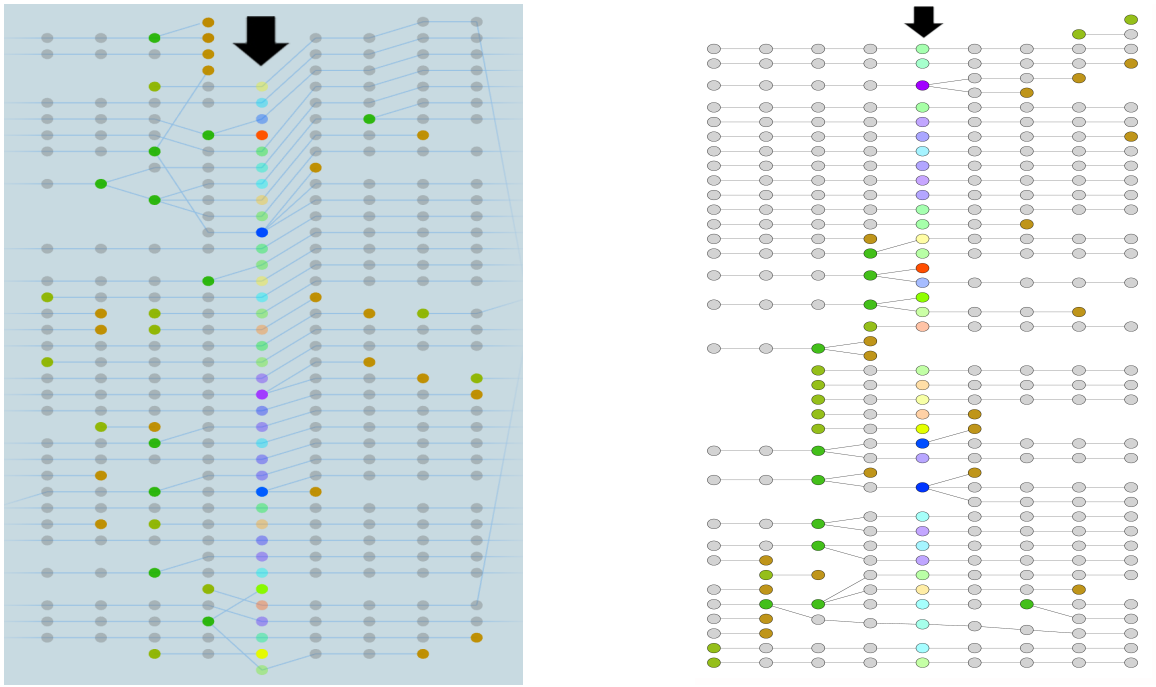
Figure 6.3: For a small tracking graph in the hydrogen flame combustion data set, graph layouts produced at each stage of the progressive two-stage graph layout algorithm. (a) Initial graph layout produced by using each time step's hierarchy layout. (b) Greedy graph layout produced using a layered optimization scheme, with the use of the median heuristic. In each graph, the focus time step is indicated with a black arrow.

difference being that in the greedy layout the newly created nodes are placed on the empty positions left by the median heuristic, whereas in the Dot layout those nodes are placed on top. More results related to this particular application example can be found in [125], [167].

6.2.2 Combustion Data - Low Swirl Flame

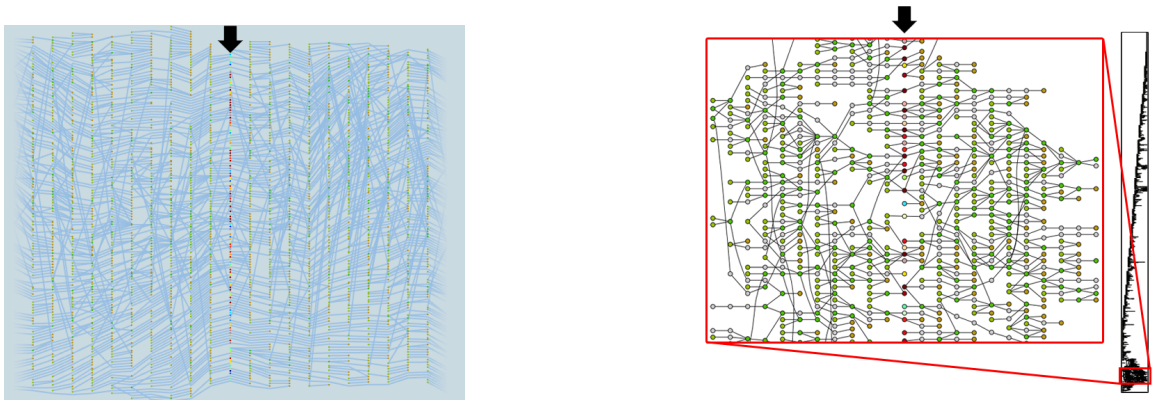
The second combustion data set represents a device scale, lean, premixed, low-swirl flame computed using a low Mach number combustion code (LMC) [181]. It consists of 331 time steps of an AMR grid at an effective resolution of $1024 \times 1024 \times 1024$, totaling about 4 TB of raw simulation data. Just as with the hydrogen flame data set, the feature of interest for this data set is also the burning cells. The clustering hierarchy of these burning cells for the fuel consumption rate is computed using threshold-based segmentation. Additionally, various feature-based attributes such as feature volume, average temperature, average H_2 consumption rate, and position are computed and stored within the hierarchy. After the preprocessing steps to create the feature hierarchies, the data size is reduced to $\approx 22GB$. Furthermore, the feature correspondences computed using a region overlap-based feature correspondence metric are stored in the meta graph structure. Those files total around 4.6GB.

Figure 6.5 shows a comparison between a layout computed in Dot and one produced



(a) Greedy graph layout (b) Dot graph layout

Figure 6.4: Comparing graphs layouts for a small tracking graph in the hydrogen flame combustion data set. (a) & (b) A comparison of the greedy layout produced by the progressive two-stage graph layout algorithm with the one produced by Dot, respectively. In each graph, the focus time step is indicated with a black arrow.



(a) Greedy graph layout (b) Dot graph layout

Figure 6.5: Comparing graphs layouts for a complex tracking graph in the Swirl flame combustion data set. (a) & (b) A comparison of the greedy layout produced by the progressive two-stage graph layout algorithm with the one produced by Dot, respectively. In each graph, the focus time step is indicated with a black arrow.

by the greedy layout. As illustrated, Dot seems unable to reuse the space occupied by dying nodes, and as a result creates a graph with extreme aspect ratios that is very difficult to explore let alone understand. The greedy layout of the proposed two-stage layout algorithm, on the other hand, manages to create a fairly compact layout with relatively few intersections. Overall, this greedy layout can produce more edge intersections, yet the resulting graph is still significantly more comprehensible. More results related to this particular application example can be found in [125].

6.3 Summary

This chapter introduces a progressive graph layout and visualization strategy that allows interactive exploration of tracking graphs, providing the flexibility to change both feature parameters and correspondence metrics on the fly. Specifically, a tracking graph is always processed with respect to a focus time step. Starting from this time step, nodes and edges are iteratively added both forward and backward in time up to a user-defined time window to form the tracking graph. As computing optimal or near-optimal layouts is expensive for larger graphs, a progressive two-stage graph layout algorithm is used to interactively visualize tracking graphs. In particular, a tracking graph is immediately visualized with a suboptimal layout that is replaced with a better one as soon as it is available. As fewer constraints must be considered, this proposed progressive graph layout and visualization strategy have the capability to produce layouts that are locally superior to a global approach. Here, several application examples from the combustion domain are also presented to demonstrate the utility of this proposed graph layout and visualization strategy.

CHAPTER 7

REDUCING VISUAL COMPLEXITY OF FEATURE EVOLUTION

This dissertation focuses on understanding dynamic data sets by exploring the evolution of their features with the aid of tracking graphs. To effectively understand the dynamic nature of data, it is essential to identify the underlying patterns and trends within data. However, for many dynamic data sets, even within a window of interest and with optimized graph layouts, the large number of features and their complex relationships make the resulting tracking graphs unmanageably large and difficult to understand. For instance, data sets often contain many spatially small features that are not necessarily of interest to the analysis. In addition, tracking graphs also contain spurious merge and split events that convey very little information and add clutter to the tracking graph. In many cases, such spurious merge and split events appear as a result of unavoidable instabilities in the parameter choices. As a result, it is not only impossible to follow a given feature through time but also difficult to identify the salient features within tracking graphs.

This chapter introduces two new strategies to reduce the visual complexity of tracking graphs. First, the definition and extraction of subgraphs is enabled to provide the flexibility to isolate interesting feature tracks and suppress small spurious structures within tracking graphs. Second, localized parameter values are used on the tracking graphs to remove artifacts that result from the effects of parameter instabilities. In particular, a novel, progressive, three-pass layout algorithm that locally adapts the parameter values within a given error bound is presented. This algorithm exploits the flexibility of defining temporally and spatially varying parameter values to produce more temporally cohesive and easier-to-comprehend tracking graphs. The specific details of these strategies are discussed in this chapter, and several application results that demonstrate their utility are also presented.

7.1 Graph Subselection

For complex tracking graphs, it is often very difficult to grasp the evolution of certain features at first glance. It can also be difficult to identify the salient features within these graphs. In such a setting, defining and extracting subgraphs in both space and time provide the flexibility to isolate interesting feature tracks. It also has the potential to suppress small spurious structures in tracking graphs. In this dissertation two approaches, filtering and feature selecting, are utilized to interactively subselect tracking graphs. In both cases, the focus is to reduce the number of nodes and edges in the graph so that its underlying patterns can be easily identifiable.

7.1.1 Filtering

Filtering is a simple but highly effective technique that can reduce the visual complexity of tracking graphs without loosing pertinent information. As previously mentioned, filtering has the capability to isolate feature tracks and suppress small spurious structures. In particular, filtering can be used to subselect graphs based on various feature-based and feature correspondence-based attributes. As discussed in Chapters 3, 4, and 5, various feature-based and feature correspondence-based attributes are precomputed within the data representations; therefore, this information can be easily used to filter tracking graphs. Even in the cases in which attribute values are not stored (e.g., the meta graph does not store feature correspondence-based attributes among features whose lifetimes do not overlap), those attribute values can be quickly computed at the tracking graph construction time.

Various feature track-based attributes such as the total amount of time of a feature's existence (i.e., track length), the maximum feature volume can also be computed at the tracking graph construction time. As a result, a tracking graph can be filtered based on its feature track-based attributes, which then subselects graphs based on the evolution of features in time. For instance, filtering tracking graphs based on the length of a track allows small spurious structures to be suppressed. Additionally, given a bounding box of interest, tracking graphs can also be filtered to spatially subselect features by screening according to the feature bounding box. Combined, these filtering options significantly reduce the visual complexity of the tracking graph by lowering both node and edge counts.

7.1.2 Feature Selection

Instead of filtering where the visual complexity of tracking graphs is reduced by screening nodes and edges based on various attribute values, sometimes it is more useful to concentrate on a particular feature and its evolution across time. For instance, dynamic data sets may contain salient features for which understanding their evolution is particularly interesting to domain scientists. Here, selecting a certain feature or a set of features in the tracking graph and then extracting all feature tracks related to those features is useful. This type of feature selection reduces the visual complexity of tracking graphs by focusing on a subset of the global tracking graph.

7.1.3 Implementation Details

As discussed in Chapter 6, tracking graph layout and visualization is deliberately designed to be conducted in a progressive fashion, and, therefore, integrating the filtering strategy is comparatively simple. Starting from a time step of interest, as nodes and their corresponding edges are progressively added to the tracking graph both forward and backward in time, those nodes and edges can be filtered according to the current filters before passing them to the layout algorithm. When integrating the feature selection strategy, a certain feature or a set of features in the focus time step can be selected, and then graph connectivity can be used to extract all related feature tracks both forward and backward in time. In each of these cases, the layout of the subgraph is computed and visualized in a progressive fashion.

7.1.4 Advantages and Limitations

Both graph subselection approaches, filtering and feature selecting, can be used to reduce the number of nodes and edges in a tracking graph. In consequence of this reduction in visual complexity within tracking graphs, the underlying patterns and trends within data can be easily identified. However, one disadvantage of these graph subselection approaches is that they do not remove the artifacts that result from the effects of parameter instabilities.

The first subselection approach, filtering, is particularly helpful as it can subselect graphs based on various attributes. Based on a single attribute value or a range of attribute values, tracking graphs can be filtered to isolate interesting feature tracks and to

suppress small structures within the graph. As most of these attributes are precomputed and stored within the flexible data representations, filtering based on various feature-based and feature correspondence-based attributes can be easily incorporated. In addition to the feature-based and feature correspondence-based filtering, feature track-based filtering can also be performed. However, it involves extra computation at the tracking graph construction time. Despite this extra computation, filtering based on feature track-based attributes has the potential to suppress small structures within the graph. Another advantage of filtering is that it can be easily integrated into the tracking graph extraction process. It fits well with the progressive graph layout and visualization strategy proposed in Chapter 6.

The second approach, feature selection, provides the means to reduce the visual complexity of tracking graphs in a different manner than filtering. This approach subselects graphs by focusing on a particular feature or a set of features and their evolution in time. This ability to extract evolution details related to a particular feature and suppress the rest of the features is one of its advantages. This approach also fits well with the progressive graph layout and visualization strategy proposed in Chapter 6. However, this type of graph subselection is effective only if the salient features can be identified and selected within the tracking graph. It might not, therefore, be the optimal way of reducing the visual complexity of tracking graphs in all dynamic data sets.

7.2 Localized Parameter Values

All feature definitions are slightly unstable (i.e., a feature may hover right at a particular parameter value). Over time, depending on whether a slightly higher or lower parameter value is selected, these features cause repeated merge and split events in the tracking graph. These types of spurious merge and split events typically convey little information of interest and clutter the view, making postprocessing of the tracking graphs unnecessarily difficult. Therefore, it is advantageous to remove such events, so the tracking graphs are more stable, easier to interpret, and better at representing the fundamental trends within the data. Filtering and graph subselection, although very helpful in isolating feature tracks and suppressing small spurious structures, do not help in removing these spurious merge and split events.

It is important to realize that choosing a single global parameter value is convenient

but by no means necessary or ideal. Additionally, it is well known that many features benefit from localized parameter values [182], [183], [62]. For example, in [183] the authors scale the parameter values by the standard deviation, and in [62] a scaling based on the local extremum is used. Furthermore, in most applications, the specific choice of parameter value is rather arbitrary, and any parameter value within some reasonable range is acceptable to the domain experts. As a result, this flexibility in choosing a parameter value is exploited, and a novel method for reducing the visual complexity of tracking graphs within a tightly controlled error bound (i.e., a reasonable range around a selected parameter value) is presented for removing the spurious merge and split events existing in the tracking graphs.

7.2.1 Progressive Adaptation of Parameter Values

This section introduces a new progressive three-pass layout algorithm that adjusts the tracking graph by locally adapting the feature parameter values within a user-defined parameter range. These local adaptations in the parameter values change the feature definitions and as a consequence remove the artifacts that arise from the effects of parameter instabilities, reducing the overall visual complexity of tracking graphs. In practice, these local changes in parameter values correspond to adjusting the cut-points within the feature hierarchies to create an arbitrary cut rather than a horizontal one (refer to Figure 3.5c in Section 3.1.2). The resulting tracking graphs are more stable, easier to interpret, and better at representing the fundamental trends within the data.

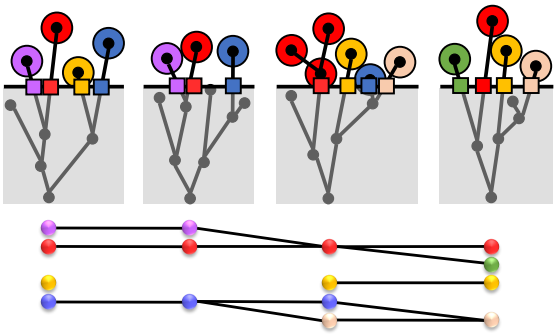
The first step toward such an algorithm is to define an objective function with respect to which the tracking graph should be optimized. In most applications, it is observed that for both postprocess analysis and visualization, domain experts prefer tracking graphs with long tracks of valence two nodes (i.e., features that evolve in isolation). Consequently, when reducing the visual complexity of tracking graphs, this adaptive algorithm focuses on reducing the total number of nonvalence two nodes. This reduction includes both splits and merges with valences greater than two as well as births and deaths with valence one. However, computing the optimal graph with the fewest nonvalence two nodes within a given parameter value range is NP-hard.

Instead, this algorithm utilizes a greedy heuristic that is efficient to implement and

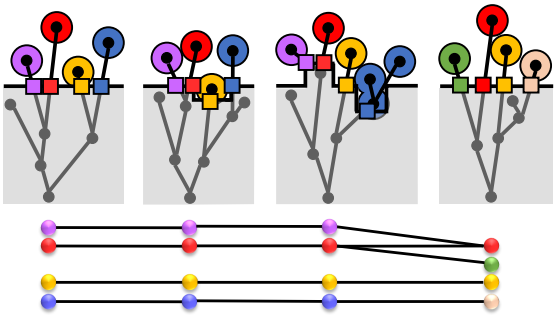
has been shown to be highly effective in practice. Given a range of acceptable parameter values or alternatively an error bound around an ideal parameter value, the tracking graph is optimized in a greedy fashion. Once the domain experts select a parameter value range, the tracking graph is optimized in three passes: left to right, right to left, and a final left to right. During each pass, as the sweep through time steps in the tracking graph is made, features in the 'previous' time step (depending on the direction of the pass, this time step could be either earlier or later in time) are always considered fixed, and the parameter values of the features in the current time step are adapted accordingly. In the first two passes, the graph is greedily adjusted to delay any event causing a nonvalence two node (i.e., merge, split, birth, and death). More specifically, when a split event is encountered, those split features are merged if it is possible within the selected parameter range. Similarly, when a merge event is encountered, if it is possible, the parameter value is adjusted to split those features.

Consider the example in Figure 7.1. The initial tracking graph, shown in Figure 7.1a, contains a blue feature that temporally splits before merging into the light pink one. The corresponding feature hierarchies for this tracking graph show that this blue feature is slowly being replaced by the light pink feature, and that the split-merge event occurs when both features exist simultaneously in the same time step. During the first pass (left to right), this split event is delayed by lowering the parameter value locally, which incidentally resolved the later merge event, creating a single clean history, see Figure 7.1b. Furthermore, the purple-red merge event is also delayed by one time step even though it is not resolved. Finally, the yellow birth and death events are also delayed and are eventually resolved. During the second pass (right to left), the remaining events from the previous pass are delayed in the other direction. As shown in Figure 7.1c, at the end of the second pass, the green split is removed, and the purple-red merge event is moved further to the left.

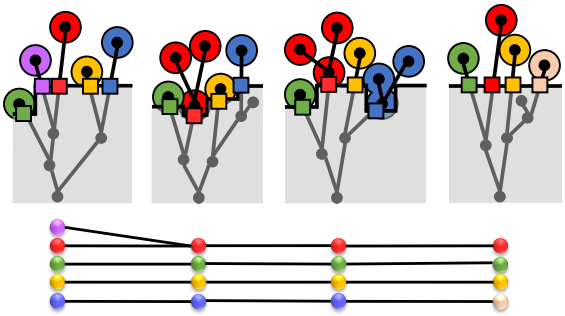
The first two passes simplify the tracking graph by locally adjusting the parameter values of the features causing a nonvalence two node. However, as a side effect, any event that cannot be resolved within the acceptable parameter value range is first moved as far right and then as far left as possible. Although these event changes are valid within the parameter value range, they do not improve the tracking graph and needlessly distort



(a) Initial tracking graph and its feature hierarchies



(b) After first pass (left to right)



(c) After second pass (right to left)

Figure 7.1: Progressive adaptation of parameter values within a tracking graph. (a) The initial tracking graph and its corresponding feature hierarchies. (b) Result after the first pass (left to right). Here, the blue split-merge and the yellow death-birth events have been resolved by slightly lowering the parameter value. The purple-red merge event has also been moved to the right by raising the parameter value, but it cannot be resolved as the purple feature disappears. (c) Result after the second pass (right to left). Here, the red-green split event has been resolved by slightly lowering the parameter value. The unresolved purple-red merge event from the previous pass is now moved to the left, yet it still cannot be resolved as the saddle necessary to affect the merge is outside the acceptable parameter value range. The third pass (which is not shown here) moves this purple-red merge event back to its original location so that the deviation from the original parameter is minimized.

the feature parameter values. Therefore, the third and final pass (left to right), rather than further simplifying the graph, attempts to correct this problem by minimizing the deviation in the parameter values. This pass moves the unresolved events back to their original parameter value. In the case of the earlier example, the unresolved purple-red merge event would be moved back to its original location in Figure 7.1a.

In practice, this parameter adaptation process is driven by the domain expert selecting a focus time step and a parameter value range. Since the parameter value for the focus time step is user-selected, it is reasonable to assume that those features at the focus time step are 'correctly' selected and can be taken as ground truths. Therefore, depending on whether the focus time step should be modified or not, two modes of tracking graph adaptation can be supported. For the first case, in which the focus time step should be modified, the entire tracking graph is processed as described above. In the second case, the parameter adaptation process proceeds simultaneously both forward and backward in time, keeping the focus time step unmodified. Instead of three passes, only two passes are required as any event that has not been resolved in the first pass cannot be resolved at all. In the case of the earlier example in Figure 7.1, the red-green split would not be resolved if the first time step were to be kept fixed. In both modes, during each pass as local, per-feature parameter values are changed, there will be features whose lifetimes do not overlap, and therefore, correspondences across those features are computed at tracking graph construction time.

7.2.2 Implementation Details

The proposed algorithm is dependent on the selection of a focus time step and a reasonable parameter value range. Once those are selected, the algorithm is implemented in a progressive manner. In particular, each pass in the algorithm sweeps through the time steps in the tracking graph. During this sweep, features in the 'previous' time step are considered fixed, and the parameter values of features in the current time step are adjusted to reduce events causing a nonvalence two node. Here, results from each pass are progressively computed, and their results are progressively visualized in the form of tracking graphs. As the processing in each pass is computed in a sweeping fashion, the results are presented progressively. Additionally, depending on whether the focus time step is modified or not, the third pass of the algorithm can be skipped.

7.2.3 Advantages and Limitations

Localized parameter values can significantly improve the tracking graphs for easier interpretation. Specifically, the proposed algorithm locally adapts the parameter values of features within a given error bound, and as a consequence reduces the artifacts that arise from the effects of parameter instabilities. Due to its greedy heuristic, it is able to present fast and effective results, and is simple and efficient to implement as well. Furthermore, the progressive nature of the algorithm makes it easy to integrate the algorithm to the tracking graph construction and visualization process. In practice, this algorithm requires only a focus time step and a parameter value range to be selected, and it also has the flexibility to fix the features within the focus time step during the parameter adaptation process.

However, approaches based on localized parameter values such as this are limited to the feature definitions given in the feature representations. In particular, the nonvalence two nodes can be reduced only if it is possible within the selected parameter value range. Furthermore, although effective, this algorithm contains a clear order dependency. Due to the greedy nature of the algorithm, each pass is optimal, but there is no guarantee that the final graph is optimal (i.e., has the fewest nonvalence two nodes). Another limitation is that reducing the number of nonvalence two nodes might not be a favorable metric across all applications. In general, this metric favors fewer features that stay stable over time and in certain cases is also prone to suppress features. As such, alternate metrics, based on which tracking graphs are optimized to reduce the effects of parameter instabilities, can be explored to improve the tracking graphs.

7.3 Application Results

As per the discussion in this chapter, this dissertation utilizes several strategies to reduce the visual complexity within tracking graphs. In the following section, the effectiveness of those strategies is demonstrated using results from several application domains.

7.3.1 Ocean Data

First, an ocean eddies data set obtained from the Los Alamos National Laboratory's Parallel Ocean Program (POP) [184] is considered. This data set contains data for the entire year of 2002 (i.e., 365 time steps) at an effective resolution of $3600 \times 2400 \times 1$, totaling about

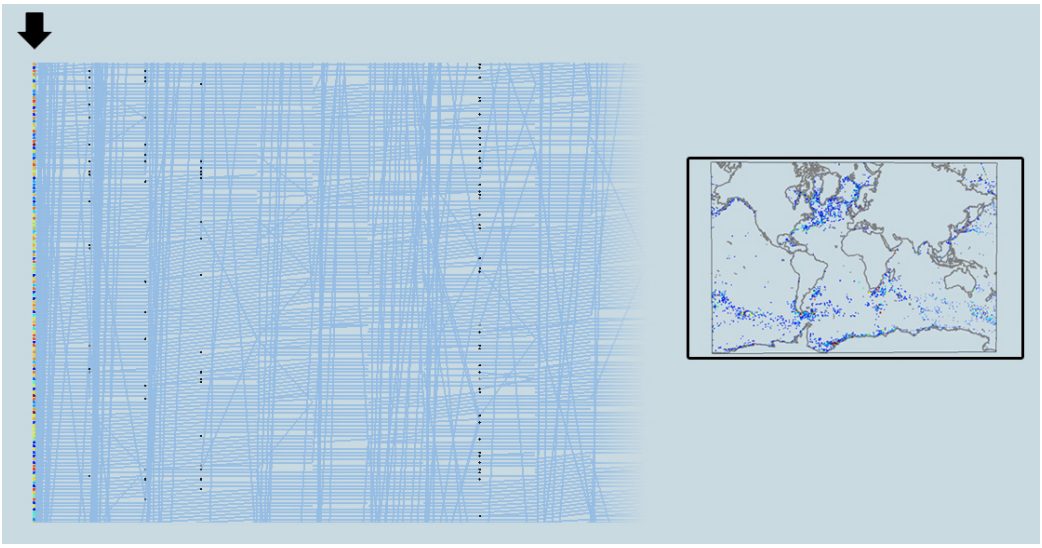
529.8GB of raw data. Although these data are originally 3D, since the z velocities in the data are negligible, they are often ignored. In this case, the top layer of data is extracted, and the evolution of ocean eddies is explored under varying Okubo-Weiss values. During a preprocessing step, first, the eddy hierarchies for a range of Okubo-Weiss values are computed, resulting in $\approx 8.4GB$ data. Along with the hierarchies, feature-based attributes such as feature volume, temperature, latitude, and longitude are stored. Then, correspondences across eddies are computed using a region overlap-based feature correspondence metric, and are stored in the meta graph structure. These meta graph files total around $\approx 4GB$.

The resulting tracking graphs for this data set contain a relatively large number of long and stable feature tracks (i.e., features that evolve in isolation), across a large range of Okubo-Weiss values. Nevertheless, due to the large overall feature count, these tracking graphs are still dense and complex. Here, the graph subselection methods discussed, specifically filtering, are of much value to domain experts. These graph subselection methods enable domain experts to explore certain regions of interest in the ocean, identify prominent eddies, and track their evolution across time. Figure 7.2 shows tracking graph results after filtering by the volume of an eddy.

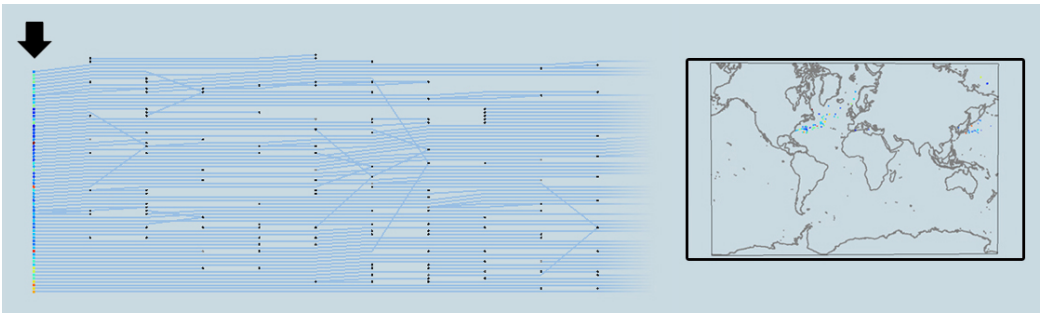
7.3.2 Atmospheric Science Data - Severe Weather Outbreak 05/2011

In this section, a second case study extracted from the same pressure-perturbation data set as in Sections 4.2.2 and 5.2.1 is considered. This pressure-perturbation data set is generated by combining high-resolution grids of numerical simulations [173] with high-frequency pressure observations [174], [175]. The case study itself demonstrates a very complex but fairly frequent meteorological situation. It took place from 0000 UTC May 24, 2011 to 0000 UTC May 26, 2011. At the beginning of this period, a discrete propagating convective system moves southeast through Oklahoma and Arkansas. Then, another propagating system forms and moves through northern Kansas from 1000-1700 UTC May 24. Beyond 2100 UTC May 24, a large-scale low-pressure system that forms over Eastern Colorado results in the development of several individual thunderstorms and mesoscale gravity waves. These individual thunderstorms then begin to congeal into many linear segments as they propagate northeast from Oklahoma and Kansas and into Missouri.

The original data for this case study contain 576 time steps and total $\approx 670MB$ of raw



(a) Original tracking graph

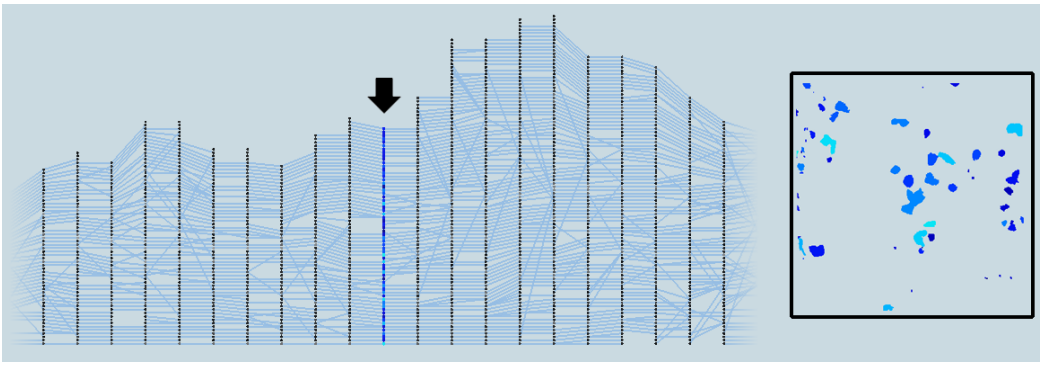


(b) Filtered tracking graph

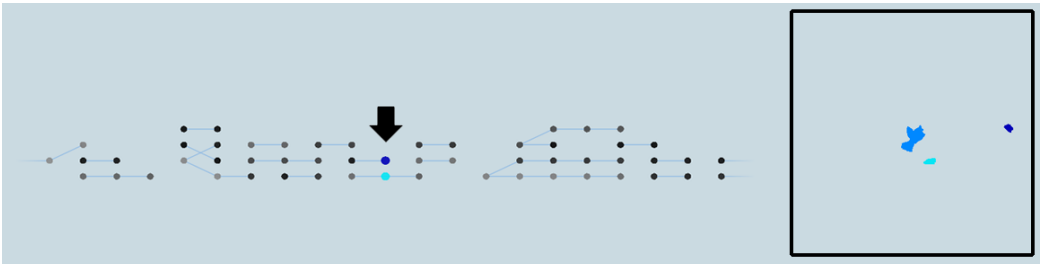
Figure 7.2: An example from the ocean eddies data set where the tracking graph in (a) has been filtered to contain only features with a volume$\geq$2000. The resulting tracking graph is shown in (b). For each tracking graph, the focus time step is indicated with a black arrow, and the features at the focus time step are visualized along each graph.

data. Once the feature hierarchies are computed using threshold-based segmentation to store the clustering hierarchy of pressure-perturbation events, and the meta graphs are computed using a region overlap-based feature correspondence metric, the total data size is reduced to around 400MB.

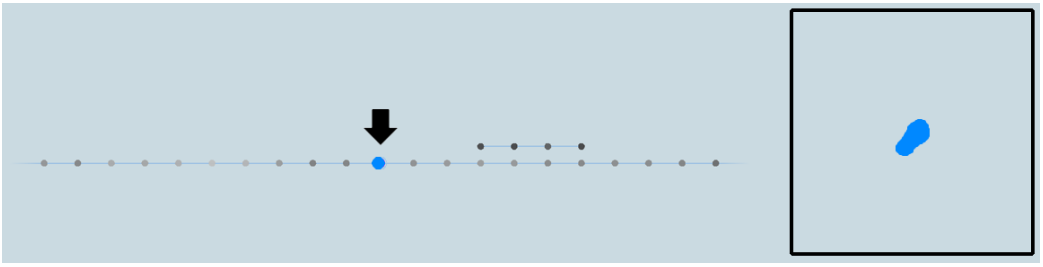
Furthermore, the ability to filter tracking graphs based on various attributes is particularly helpful in reducing the visual complexity of tracking graphs for this data set. In this case, filtering based on feature track-based attributes (which are computed at tracking graph construction time), specifically filtering based on the total propagation distance of a feature, reveals interesting insights about this data set. For the tracking graph in Figure 7.3a, the total propagation distance of a feature is used to filter the tracking graph



(a) Tracking graph for the original data set



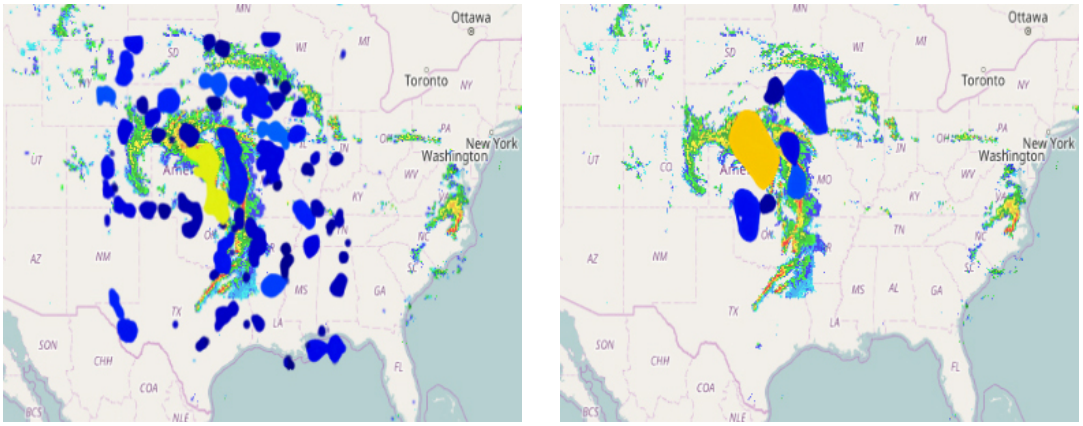
(b) Graph in (a) filtered by total propagation distance of a feature



(c) Tracking graph for the cleaner, better preprocessed data set

Figure 7.3: For the atmospheric science case study, comparison of tracking graphs extracted for the two data sets. (a) Tracking graph constructed from the original data set. (b) Tracking graph after filtering the graph in (a) by total propagation distance of a feature ($\geq$10). (c) For the same focus time window, its equivalent tracking graph for the better preprocessed data set. In each graph, the focus time step is indicated with a black arrow, and the features at the focus time step are visualized along each graph.

and retain those features that are nonstationary, see Figure 7.3b. Comparing these filtered features against the radar imagery indicates the existence of noise within the data set, see Figure 7.4. Specifically, those filtered, stationary features should be considered background noise and need to be ignored in the analysis. This new insight regarding 'stationary' noise within the data has inspired the collaborating scientists to improve their preprocessing process to remove such background noise more effectively. Consequently, the atmospheric scientists are able to generate a cleaner data set totaling $\approx 1.1GB$ for the same case study.



(a) Original data set (b) Cleaner, better preprocessed data set

Figure 7.4: For the atmospheric science case study, features existing at 1.0 hPa for the same time step (at $T = 326$) in both (a) the original data set and (b) the cleaner, better preprocessed data set are visualized alongside their radar imagery. As indicated in the images, the preprocessed data set (b) contains less noise than the original data set (a).

For this new data set, the total data size for the feature hierarchies and meta graphs is around 80MB. Figure 7.3c shows an equivalent but cleaner tracking graph of Figure 7.3a for this new data set. When exploring this new data set, the tracking graphs clearly reveal the complexity of the case study, with large and complex graph structures, see Figure 7.5. More results related to this particular application example can be found in [176].

7.3.3 Combustion Data - Hydrogen Flame with No Turbulence

In this section, the same combustion data set in Sections 3.2.2 and 6.2.1 is considered. The feature hierarchies for this data set store the clustering hierarchy of burning cells and are computed using threshold-based segmentation, and the meta graph structure is computed using a region overlap-based feature correspondence metric. The original data set totals around 400GB. After the preprocessing steps to create the feature hierarchies and meta graph, the data size is reduced to ≈800MB.

Figure 7.6 shows several examples from this data set, where graphs have been subselected based on filtering and feature selection. Both of these graph subselection options have the capability to isolate interesting feature tracks within tracking graphs. Specifically, filtering by the values of attributes or based on a bounding box of interest reduces the visual complexity of tracking graphs by screening nodes and edges, whereas feature

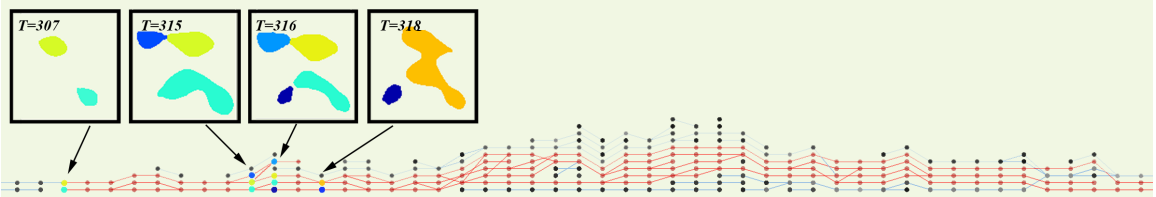
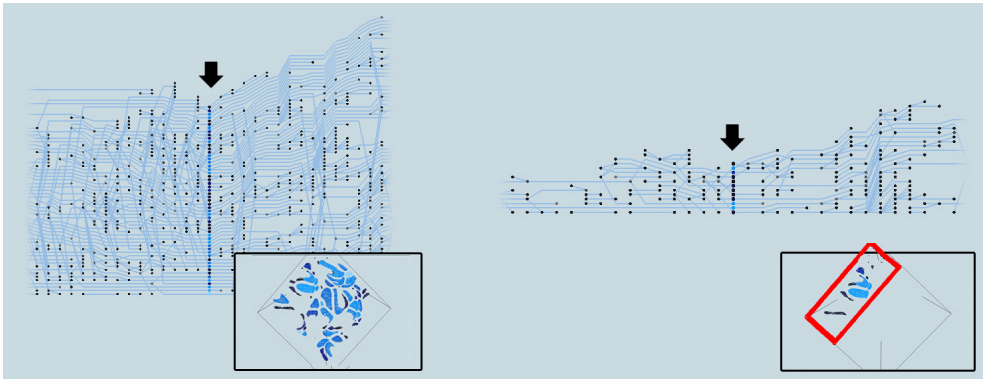
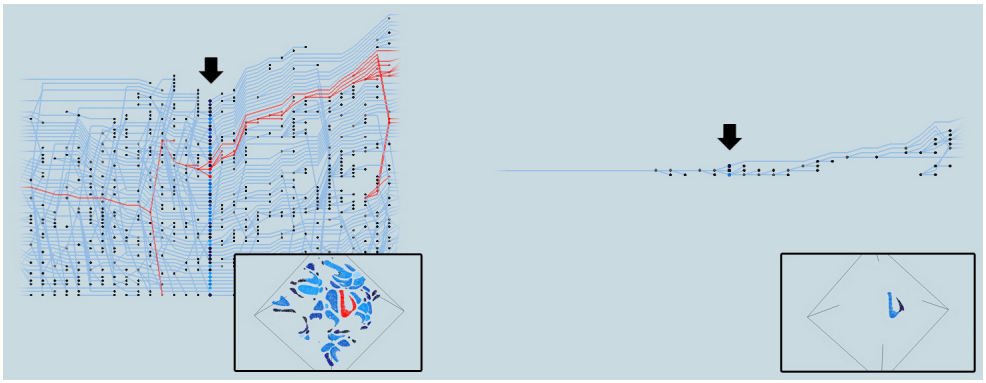


Figure 7.5: For the better preprocessed data set in the atmospheric science case study, a longer tracking graph showing the evolution of pressure-perturbation events for 49 time steps at 1.0 hPa. For the two selected feature tracks in red, the features existing at $T = 307$, $T = 315$, $T = 316$, and $T = 318$ time steps are visualized. This graph clearly shows the complexity within this case study. Several thunderstorm complexes appear to merge, split, form, and dissipate.



(a) Subgraphs in space



(b) Subgraphs relative to a feature of interest

Figure 7.6: For the hydrogen flame combustion data set, extracting local graphs either in (a) space or (b) relative to a feature of interest. In each case, the tracking graphs both before (left) and after (right) the selection are shown. In each graph, the focus time step is indicated with a black arrow. For additional details, the features at the focus time step are displayed alongside the tracking graphs. In those features, the bounding box of interest in (a) and the selected feature of interest and its track in (b) are indicated in red.

selection can focus on a particular feature or a set of features and their evolution across time.

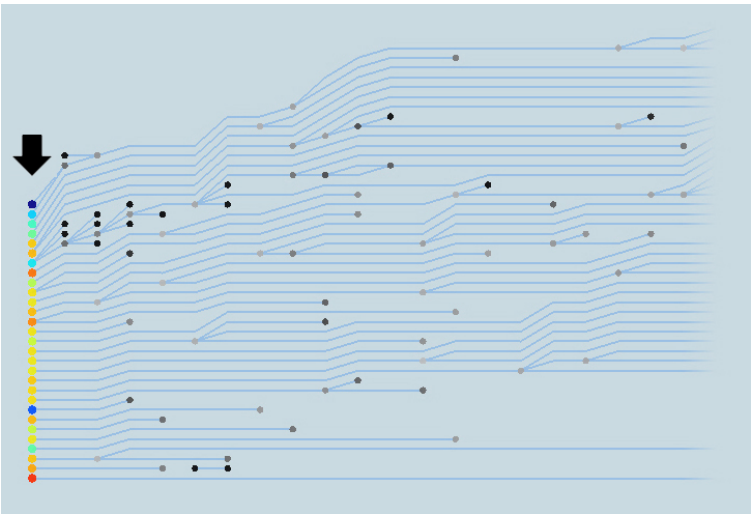
Another example from this data set, where the visual complexity of tracking graph has been reduced using the proposed three-pass layout algorithm, is shown in Figure 7.7. The localized parameter values within a complex tracking graph have been modified within a reasonable parameter range to produce cleaner, easier-to-interpret tracking graphs. Several other results related to this application example can be found at [125], [167].

7.3.4 Combustion Data - Turbulent Counterflow Flame

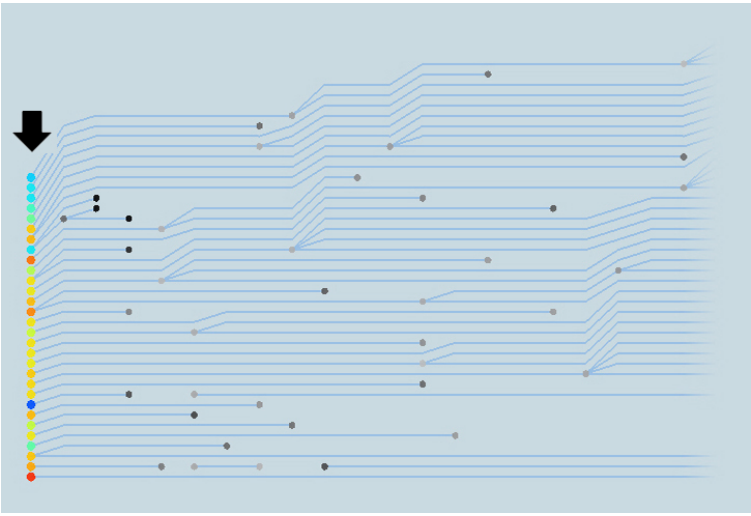
As a second combustion data set, a data set from a simulation of a flame in a highly turbulent premixed counterflow is considered. The original data for this simulation have a resolution of $432 \times 640 \times 640$ at 1048 time steps and total around 2.7TB. The extinction regions in the flame are considered to be the feature of interest for this data set, and understanding how these extinction regions evolve across time is of utmost importance to the collaborating scientists. To this end, the clustering hierarchy of extinction regions for a range of parameter values is computed using a threshold-based segmentation. The feature-based attributes such as feature curvature, area, circumference, normal strain, and tangential strain are computed and stored within the hierarchy.

Due to the large data sizes within the original data set (2.7TB), the preprocessing steps to create these feature hierarchies have been performed at NERSC on the Carver cluster with 16, 2.67 GHz Intel Xeon processors per node. The resulting data size is $\approx 670MB$. For this simulation, the flame surface is rather stationary, so a simple distance-based feature correspondence metric is used for tracking these extinction regions across time. The feature correspondences computed are stored in the meta graph structure, and those files total around 63MB.

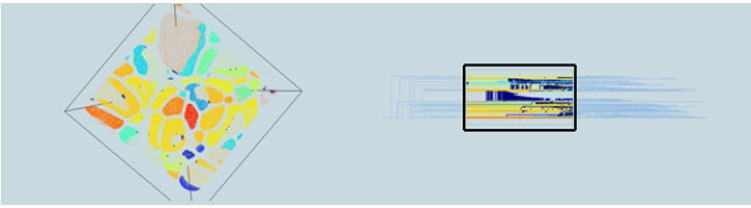
This data set contains far fewer features but due to the nature of the data, analysis is more parameter unstable. In other words, features contain unstable parameters, making it hard to select a fixed parameter value on which to explore feature evolution. Here, the proposed three-pass layout algorithm that removes artifacts that result from the effects of parameter instabilities by utilizing localized parameter values is of much value. Figure 7.8 shows an example from this data set, where the localized parameter values have been



(a) Original tracking graph



(b) Simplified tracking graph



(c) Features (left) and the user-defined parameter range (right)

Figure 7.7: An example from the hydrogen flame combustion data set where the tracking graph in (a) has been simplified within the user-defined parameter range to produce the simpler tracking graph in (b). In each tracking graph, the focus time step is indicated with a black arrow. For the graph in (a), both features (left) and the user-defined parameter range (right) at the focus time step are displayed in (c).

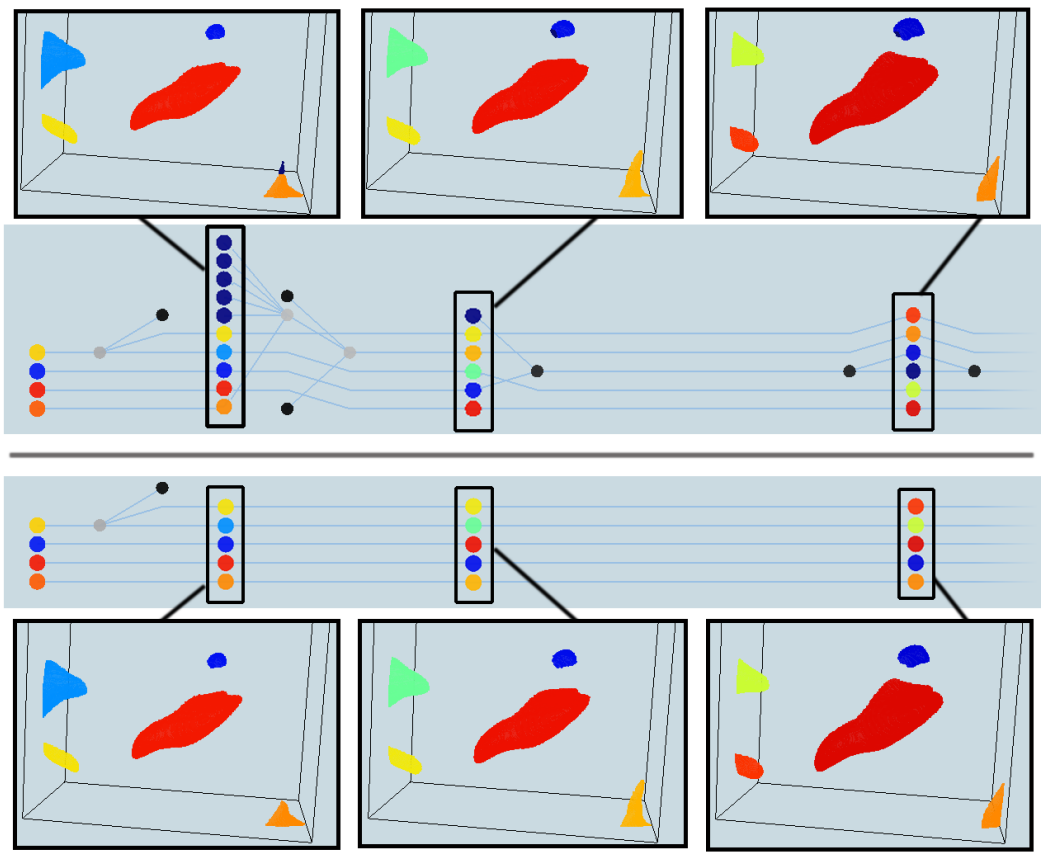


Figure 7.8: A tracking graph simplification example from the counterflow combustion data set. The features and the tracking graph both before (top) and after (bottom) the parameter adaptation process are displayed. Most of the simplified features are too small to see at this scale, and thus are insignificant, but without localized parameter values, those features create a substantially more complex tracking graph.

used to extract cleaner tracking graphs for visualization and analysis. More results related to this particular application example can be found in [167].

7.4 Summary

Data size, spatially small features, artifacts, and noise within data can cause tracking graphs to become nearly unmanageably large and difficult to understand. This chapter discusses novel strategies to reduce the visual complexity within tracking graphs. Specifically, using two graph subselection approaches, filtering and feature selecting, subgraphs are defined and extracted. These approaches provide the flexibility to isolate interesting feature tracks and suppress small spurious structures in tracking graphs. Furthermore, a new progressive three-pass layout algorithm that can reduce the artifacts resulting from

effects of parameter instabilities is also introduced. This chapter presents the algorithm and other implementation details related to these proposed strategies. Finally, several application examples from ocean science, atmospheric science, and combustion domains are presented to demonstrate the utility of the proposed strategies.

PART IV

EXPLORING FEATURE EVOLUTION VIA A GENERIC FRAMEWORK

CHAPTER 8

INTERACTIVE EXPLORATION AND ANALYSIS OF FEATURE EVOLUTION

For understanding dynamic data sets via exploring the evolution of features, apart from methods for encoding and extracting features and their correspondence details, it is also essential to maintain an interactive visualization and analysis environment. As discussed in Section 1.2.3, a wide variety of approaches exist for the purpose of studying the dynamic aspects of data sets. However, it is often the case that most of these approaches are custom tailored to their specific use cases. With respect to the general notion of dynamic features, this specialization aspect is rather arbitrary and unnecessary. Instead, rather than focusing on one specific domain, a generic approach that is applicable across a variety of application domains, is more appropriate, and has the potential for understanding feature evolution across time in a more general fashion. This chapter discusses different aspects of a novel generic framework intended for exploring feature evolution across time and presents several application examples to demonstrate its generality.

8.1 Generic Framework for Exploring Feature Evolution

This section provides a comprehensive description of a generic framework for exploring feature evolution across time, partitioned into several subsections dealing with design, visualization, exploration, and implementation. Here, the generalization of this framework is achieved through abstraction. Given the appropriate abstractions for feature definitions and their correspondences, as discussed in Chapters 3, 4, and 5, efficient data structures and visualization techniques are integrated to design a system for addressing the general task of understanding feature evolution across time. If necessary, this proposed generic framework also has leeway for including additional domain specific functionality to better facilitate the requirements of scientists.

8.1.1 Framework Design

This proposed generic framework consists of three views: feature hierarchy, feature evolution, and data embedding. The first two views present general conceptual views of the time-dependent feature hierarchies, whereas the third presents a view for the geometric embedding of features, see Figure 8.1. Combined, these views allow users to explore features and their evolution and how different scales affect their behavior.

8.1.1.1 Graph Display for Feature Hierarchy

As the name suggests, this component is dedicated to visualizing the feature hierarchies of each time step. For a particular data set, as discussed in Chapters 3, 4, and 5, its hierarchical feature representations can result in either a tree or a graph. In this view, a feature hierarchy in either of these forms is visualized via a hierarchical graph layout. For a given data set, this hierarchical graph layout of each feature hierarchy follows the depth-first ordering of its features and is computed only once as the data are read for the first time. Specifically, the feature hierarchy for a user-defined focus time stop is visualized

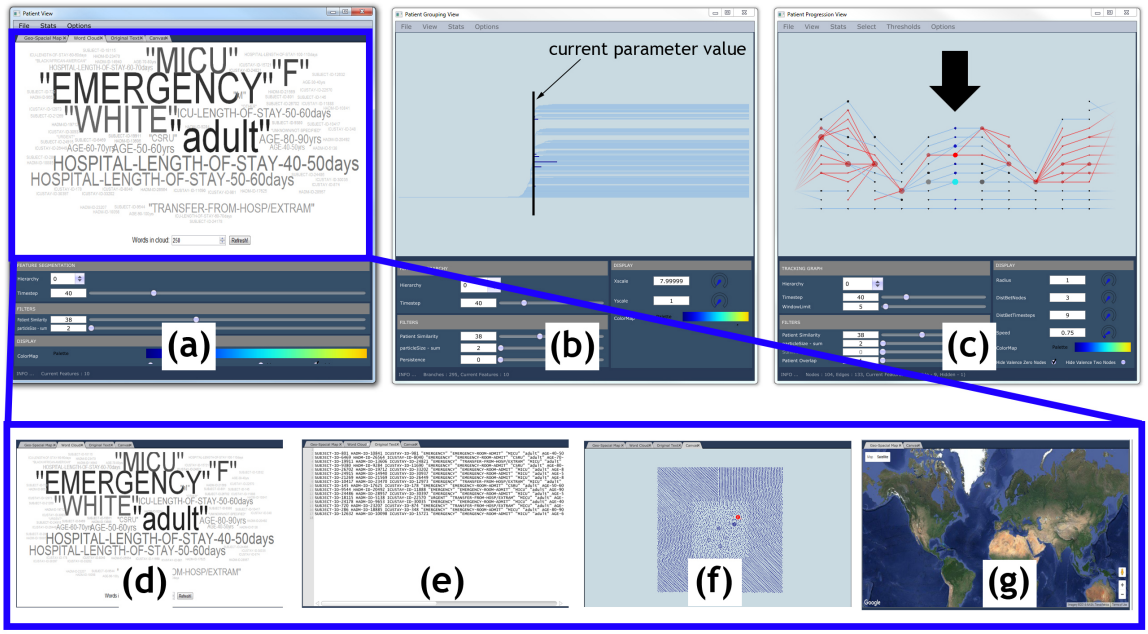


Figure 8.1: The system contains a view for (a) data embedding, (b) feature hierarchy, and (c) feature evolution. The data embedding view consists of several subcomponents: (d) a word cloud, (e) textual, (f) geometric embedding, and (g) geospatial views. Here, a healthcare data set is used, and a selected feature and its track are indicated in red.

in this view. When the focus time step is changed, the new feature hierarchy is visualized using the computed graph layout, and as the scale changes its active features are highlighted. As shown in Figure 8.1b, the selected parameter value for the full parameter range is displayed in a black vertical line, and the active features are highlighted in prominent colors. Additionally, for flexibility, the feature hierarchy is allowed to be visualized either horizontally or vertically. As the number of features within a time step increases, their feature hierarchies also become complex and difficult to understand. Therefore, this view allows visualizing certain slabs within the feature hierarchy. For example, as displayed in Figure 8.2, when users define a fuzzy parameter region for simplifying the tracking graphs (for the three-pass layout algorithm discussed in Section 7.2.1), the feature hierarchy slabs for those fuzzy regions are visualized along with its range of outcomes.

8.1.1.2 Graph Display for Feature Evolution

In this view, the evolution of features is visualized with the use of tracking graphs. A tracking graph looks similar in nature to parallel coordinates and displays a collection of feature tracks. When visualizing tracking graphs, this view always processes a tracking graph with respect to a focus time step. Starting from the focus time step, nodes and edges are iteratively added to the tracking graph in both forward and backward in time up to a user-defined time window, see Figure 8.1c. Each set of nodes in the same x coordinate

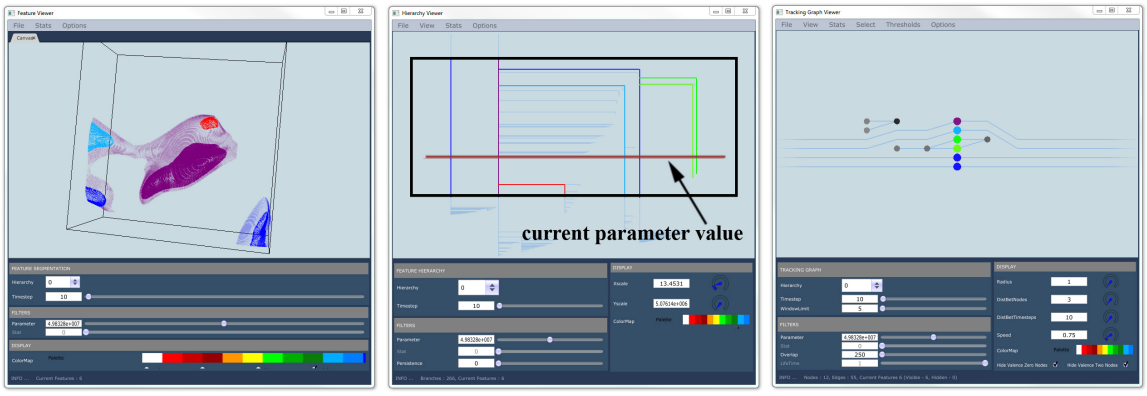


Figure 8.2: In the system, when using the three-pass layout algorithm discussed in Section 7.2.1, the range of potential outcomes for the user-defined parameter range of the focus time step is displayed for both feature hierarchy (middle) and data embedding views (left). Here, a combustion data set has been used and the user-defined fuzzy region is outlined in 'black'. The feature hierarchy is displayed using vertical layout.

indicates features in one time step, and edges across them indicate their correspondences. For visual clarity, the set of nodes in the focus time step is always displayed in prominent colors. Furthermore, progressive techniques are used to both interactively lay out and visualize these tracking graphs. Specifically, a fast initial graph layout and a slower greedy one are employed as discussed in Chapter 6. As a result, users are immediately presented with an initial graph layout that is replaced by a better one (with fewer edge intersections) as soon as it is available. This initial graph layout is computed by obtaining feature positions within the respective feature hierarchy layouts (which are computed for the previous component), and the greedy layout is computed by using a layered optimization scheme, specifically, the median heuristic, which moves outward from the focus time step. One of the main advantages of this progressive strategy is that it preserves the interactive nature of the process – the user will always see some information very quickly – even for large graphs and long time windows.

8.1.1.3 Display for Data Embedding

Visualization techniques from SciVis and InfoVis domains are combined here to present a specialized view for data embedding, see Figure 8.1a. Specifically, as shown in Figure 8.1d–g, visualization techniques such as geometric embedding, geospatial, word cloud, and textual visualizations are integrated within this view.

8.1.1.3.1 Geometric visualization. Regardless of the data type and domain, visualizing the geometric embedding of features can reveal interesting details and trends. This component visualizes the geometric embedding of data in either 2D or 3D, see Figure 8.1f.

8.1.1.3.2 Geospatial visualization. When relevant information is available, feature details are augmented with geospatial visualizations (e.g., physical location of an eddy, geographic location of where a tweet was posted), see Figure 8.1g.

8.1.1.3.3 Word cloud visualization. This component is dedicated to providing a quick overview of features with textual information using word clouds. For a selected feature, a word cloud is constructed from its textual data, as shown in Figure 8.1d.

8.1.1.3.4 Textual visualization. This component visualizes textual data in their native domain (i.e., as text), see Figure 8.1e. However, this component is not limited to textual data and can be used to display any feature-based attribute with textual information.

8.1.2 Data Exploration

The generic framework described in this chapter makes use of progressive visualization techniques for interactivity. Given a dynamic data set, data are always presented with respect to a focus time step that is processed first. Subsequently, data are extracted and presented for neighboring time steps in the order of increasing distance. Furthermore, all views designed for only a single time step (i.e., feature hierarchy view and data embedding view) use the focus time step to determine their time step. The feature evolution view, starting from the focus time step, iteratively adds nodes and edges to the tracking graph both forward and backward in time up to a user-defined time window.

Parameters, such as feature definition, correspondence, and filtering, are coordinated across all three views to provide a fully linked analysis environment. Through this linked-view interface, users are allowed to explore each data set by changing the focus time step and time window. To help users maintain context across multiple views, correlated color maps are utilized, and features are allowed to be colored using various feature-based attributes such as volume, area, and curvature. Additionally, parameters that define the feature hierarchy, correspondence, and other feature-based attribute values (e.g., volume, position, area, and temperature) can all be explored. In addition to these parameters, which are common across all views, several view-specific parameters are also included within the system. For example, the tracking graphs can be filtered by various feature track-based attributes, and the feature hierarchies can be filtered by the persistence of a certain feature defined to be the length from the branch that represents that feature to its leafs (i.e., absolute function value difference). Furthermore, to prevent visual clutter within the tracking graph, valence two and zero nodes can be hidden, nodes can be scaled based on their volume, and important events such as feature births, deaths, splits, and merges can be highlighted.

8.1.3 Implementation Details

The interactive visualization and analysis environment outlined in this chapter is implemented using the ViSUS framework [185], [186], which provides the basic building blocks for designing a streaming, asynchronous data flow. As shown in Figure 8.3, this proposed generic framework's data flow consists of several modules.

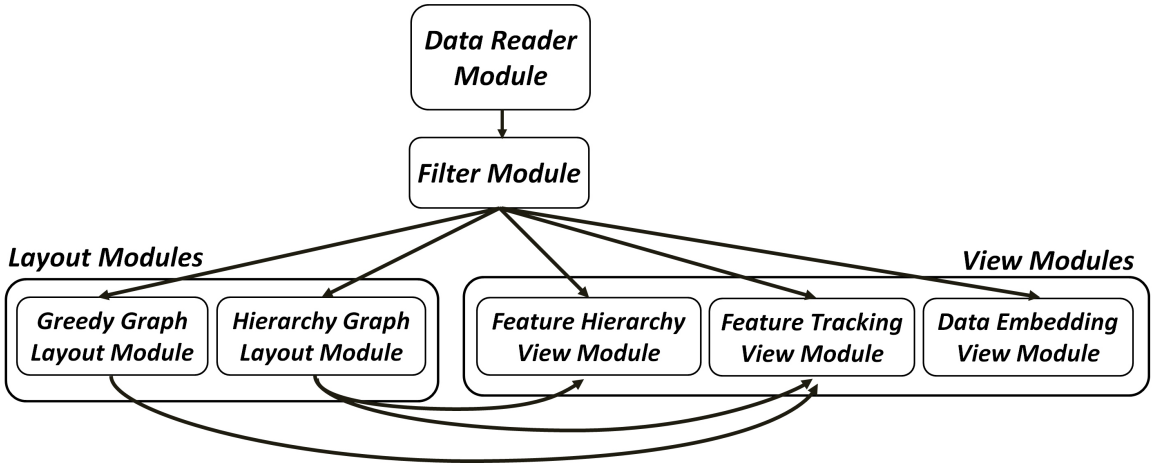


Figure 8.3: The core data flow of the proposed generic framework. `Data Reader` reads data and pass them to the `Filter` where relevant nodes and edges are extracted according to the current parameters. Then, the resultant data are simultaneously sent to the `Layout` and `View` modules. The `Layout` modules compute the relevant graph layouts and send them to the `View` modules for rendering.

The `Data Reader` module loads requested data and passes the corresponding node and edge information (i.e., features and their correspondences) to the `Filter` module. The `Filter` module filters received information to extract the nodes and edges for the current parameter and attributes values (e.g., volume, area, feature correspondence metric, and track length) and simultaneously sends the results to the layout and view modules.

The two layout modules compute the corresponding tracking graph layouts and send them to the relevant view modules for rendering. Given a data set, the `Hierarchy Graph Layout` module computes the graph layouts for the feature hierarchies (these are computed once as the data are first read into the system), and then sends those details to the `Feature Hierarchy View` and `Feature Tracking View` modules. The `Greedy Graph Layout` module computes the greedy layout for the tracking graph each time its parameters change and passes that layout to the `Feature Tracking View` module.

As the name suggests, the `Feature Hierarchy View` module visualizes the focus time step's feature hierarchy. The `Feature Tracking View` module first renders the tracking graph using the initial graph layout computed based on their feature hierarchies. Once the greedy layout becomes available, it is integrated with the current graph. Finally, the third view module, `Feature Embedding View`, displays features embedded in the domain.

Each time a user changes parameters and/or selections, current processing within the data flow is interrupted and restarted. However, rendering in the views maintains its current state for visual continuity. Furthermore, each parameter change is communicated across all modules through the linked-view interface for consistency.

Most parts of the system (all except word cloud and geospatial visualizations) are implemented in C++ and use OpenGL rendering. The word cloud and geospatial visualizations make use of JavaScript libraries and functions. Specifically, d3-cloud [187], a Wordle-inspired word cloud layout; OpenLayers [188], a library for interactive maps; and Google maps [189] are used. Within the system, the integration between C++ and JavaScript is achieved using the Awesomium library [190], which enables C++ code to be seamlessly integrated with HTML UI and to maintain interactions across the two.

8.1.4 Advantages and Limitations

This generic framework has a significant advantage compared to other existing approaches, especially due to the generality of its design. First, as discussed in Chapters 3, 4, and 5, all methods used for defining and tracking features are designed to be general. Any of the standard clustering algorithms can be used for defining hierarchical features, and any existing feature correspondence criteria can be used to define feature correspondences. Second, the various layout, visualization, and simplification techniques introduced, as discussed in Chapters 6 and 7, together with the proposed visualization and analysis environment, provide the capability to interactively explore the evolution of features within dynamic data sets. Consequently, various large-scale scientific and nonscientific dynamic data sets can be explored within the proposed generic framework to understand their underlying dynamic nature. Finally, in addition to the existing functionality within the system, it also has leeway to include additional domain-specific functionality to better facilitate the requirements of scientists.

8.2 Application Results

As a part of the research work that has been carried out during this dissertation, a variety of dynamic data sets have been used in this proposed visualization and analysis environment. Specifically, combustion, ocean science, cosmology, plasma-surface interac-

tions, and atmospheric science data sets in the scientific domain and social media, healthcare, and image data sets in the nonscientific domain have been successfully used in this framework to explore their dynamic nature across time, see Figure 8.4. Together, these data sets demonstrate the generality and applicability of this framework for exploring the evolution of features across time, see Table 8.1. A list of different classifications compiled for these data sets is presented in Table 8.2.

In this section, two real-word examples from social media and atmospheric science domains are presented to demonstrate the effectiveness and generality of the proposed generic framework. The example from the social media domain makes use of the existing functionality of the proposed generic framework, whereas in the example from the atmospheric science domain, several new functionalities are added to the system for better facilitating the requirements of atmospheric scientists.

8.2.1 Social Media Data

In this dissertation, a Twitter data set that has been collected by the Twitter streaming API for 5 months, beginning from February 2013 (i.e., 150 time steps), is used for exploring

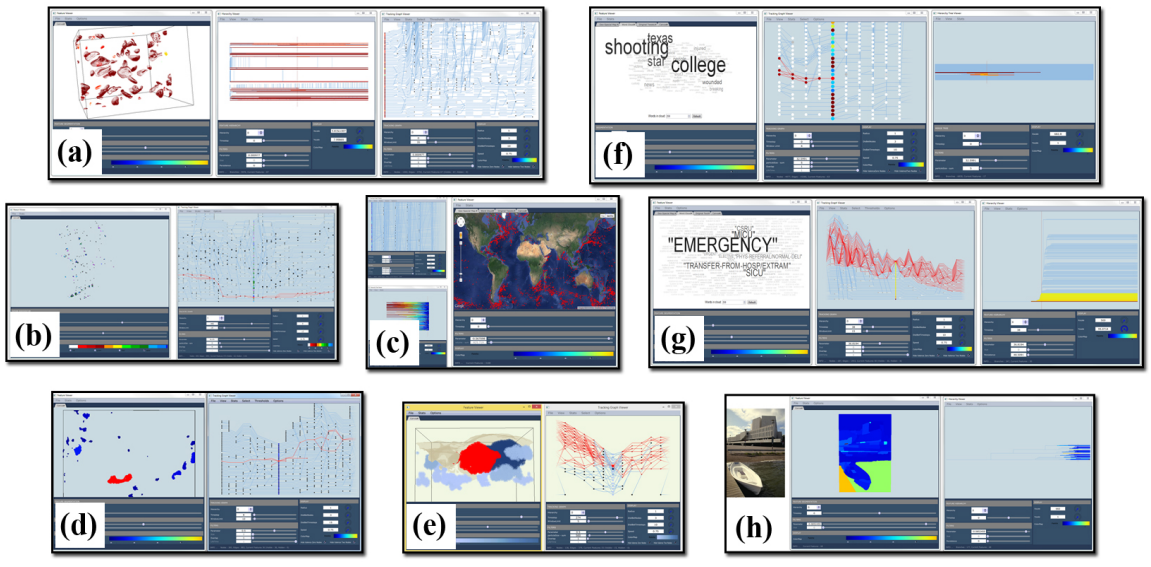


Figure 8.4: A variety of dynamic data sets have been explored within the proposed generic framework to demonstrate its generality: (a) combustion, (b) cosmology, (c) ocean science, (d) atmospheric science, (e) plasma-surface interactions, (f) social media, (g) healthcare, and (h) image.

Table 8.1: Data sets explored so far within the proposed generic framework. In each case, their feature of interest, feature hierarchy and feature correspondence details are presented.

Data set	Feature Hierarchy	Feature Correspondence
Combustion	Using topology-based segmentation, the hierarchy of burning cells for a range of fuel consumption rates	Feature intersection-based (region overlap), distance-based
Ocean Science	Using topology-based segmentation, the hierarchy of ocean eddies for a range of Okubo-Weiss values	Feature intersection-based (region overlap)
Cosmology	Using distance-based clustering, the hierarchy of halos for a range of distance (linking length) values	Feature intersection-based (region overlap)
Plasma-surface Interactions	Using distance-based clustering, the hierarchy of helium bubbles for a range of distance values	Feature intersection-based (region overlap)
Atmospheric Science	Using topology-based segmentation, the hierarchy of storm systems for a range of pressure values	Distance-based
Social Media	Using similarity-based clustering, the hierarchy of twitter topics for a range of textual similarity values	Feature intersection-based (element overlap)
Healthcare	Using similarity-based clustering, the hierarchy of patient groups for a range of patient similarity values	Feature intersection-based (element overlap)
Image	Using topology-based segmentation, the hierarchy of image regions for a range of region saliency values	Feature intersection-based (region overlap)

the progression of Twitter topics across time. Specifically, two sets of tweets extracted from this Twitter data set are considered: one consisting of ≈7 million tweets from organizations (≈54,000 tweets per day, totaling about 1GB of raw data) and the second consisting of ≈5 million individual tweets (≈26,000 tweets per day, totaling about 700MB of raw data). In addition to exploring the progression of tweets in general, the domain scientists are particularly interested in finding the differences between these two sets of tweets.

First, an offline preprocessing step is used to compute the feature hierarchies and meta graph for both data sets. A connectivity-based clustering algorithm that considers bi-gram similarity is used for computing the clustering hierarchy of tweets, and the same bi-gram metric is used for feature correspondence computation. Using these precomputed data

Table 8.2: Several different classifications for the data sets explored within the proposed generic framework.

By application domain	*Scientific* - combustion, ocean science, cosmology, plasma-surface interactions, atmospheric science *Nonscientific* - social media, healthcare, image
By data type	*Structured grid* - combustion, ocean science, atmospheric science, image *Point set* - cosmology, plasma-surface interactions *Text* - social media, healthcare
By feature grouping algorithm	*Topology-based segmentation* - combustion, ocean science, atmospheric science, image *Clustering* - cosmology, plasma-surface interactions, social media, healthcare
By feature tracking metric	*Feature intersection-based* - combustion, ocean science, cosmology, plasma-surface interactions, social media, healthcare, image *Distance-based* - atmospheric science, combustion

representations and the proposed visualization and analysis framework, scientists are allowed to understand how Twitter topics split, merge, and relate to other topics within a time step, and also explore the progression of Twitter topics under varying text similarities. The geometric embedding view within the system is very helpful to gain a quick overview of the clustering relationships among Twitter topics. However, as the geometric embedding is not so obvious for Twitter data, the GraphViz [191] 'neato' layout algorithm together with bi-gram similarities is used to compute a 2D embedding for Twitter topics. Figure 8.5f shows the geometric embedding of Tweets for a specific time step within the organization tweets data set. Moreover, the geospatial view within the proposed generic system is able to provide insights into the relationships between certain Twitter topics and a geographic location, see Figure 8.5d.

For both data sets, even with the simple bi-gram similarity metric, the feature hierarchies and their correspondences are able to produce sensible feature tracks that connect related tweets. For example, feature tracks regarding various incidents of the year 2013 (e.g., oil spill liability cap issue, NASA's Curiosity rover's activities on Mars, Paris Jackson's suicide attempt) could be explored. Figure 8.5 shows a snapshot of the proposed generic system while exploring the feature track for the oil spill liability cap issue.

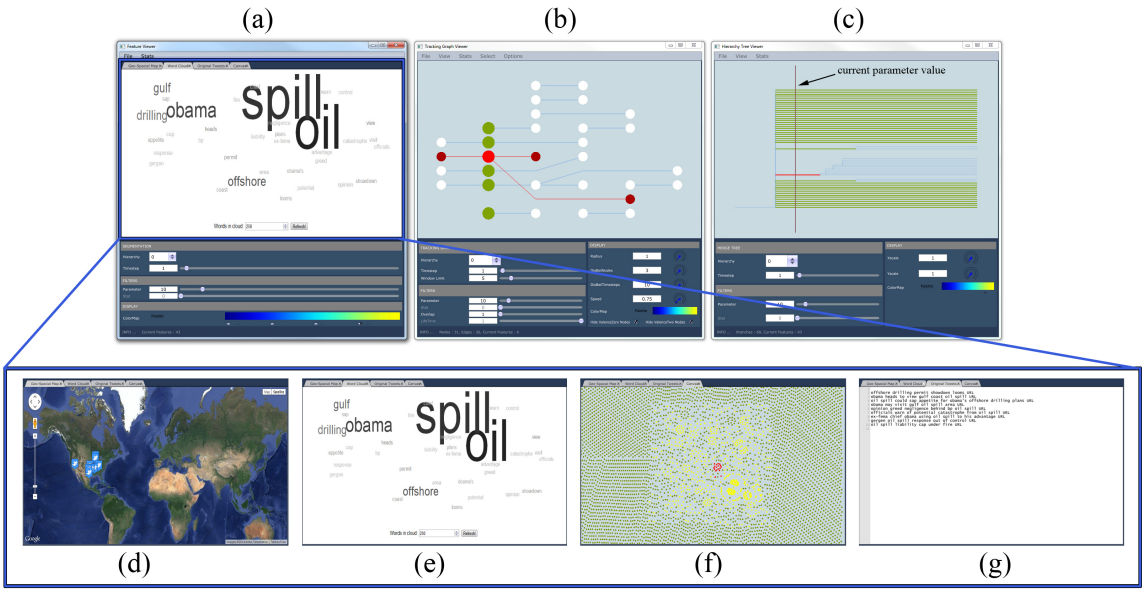


Figure 8.5: The proposed system visualizing social media data. The (a) data embedding, (b) feature evolution, and (c) feature hierarchy views are displayed. For the data embedding view, its subcomponents, (d) a geospatial, (e) word cloud, (f) geometric embedding, and (g) textual views, are also displayed. The organization tweets data set is used, and a selected feature regarding the 2013 oil spill liability cap event and its track are indicated in red.

Comparing both data sets provides valuable insights. The first insight is that the types of tweets posted by these two user groups are drastically different in their content. Specifically, the organizational tweets carry substantial amounts of event information that interests the public, whereas the individual tweets are mostly about day-to-day activities of ordinary individuals, see Figure 8.6. The second insight is that for generic topics (e.g., spring break, birthdays, and Valentine's Day), the individual tweets are a good source to explore the points of views of a large number of people. Third, since individuals mostly use hand-held device apps (e.g., Twitter for iPhone) to author their tweets, compared to the tweets of organizations, individual tweets contain more geospatial information.

8.2.2 Atmospheric Science Data

Many weather events are characterized by variations in surface pressure from the mean pressure value (i.e., pressure-perturbations). Atmospheric scientists are very much interested in using surface pressure for characterizing a wide variety of weather events, and then exploring those pressure-perturbations both in space and across time to better

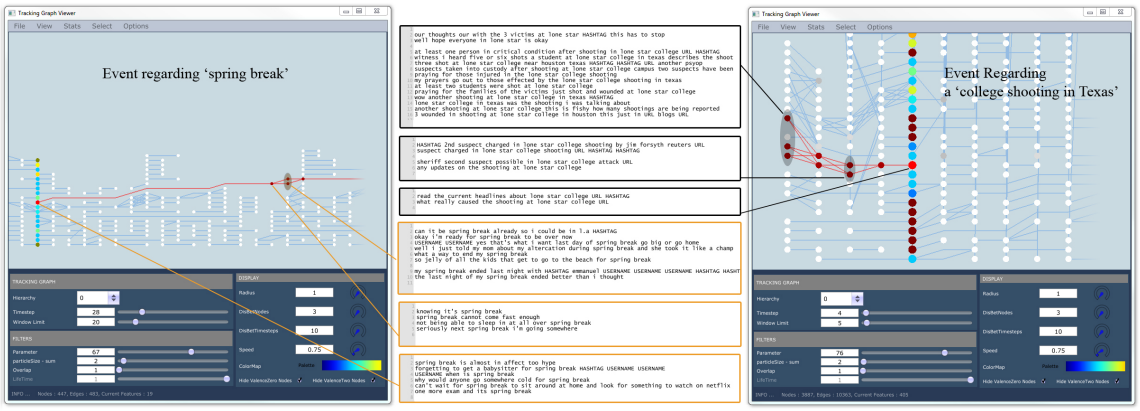
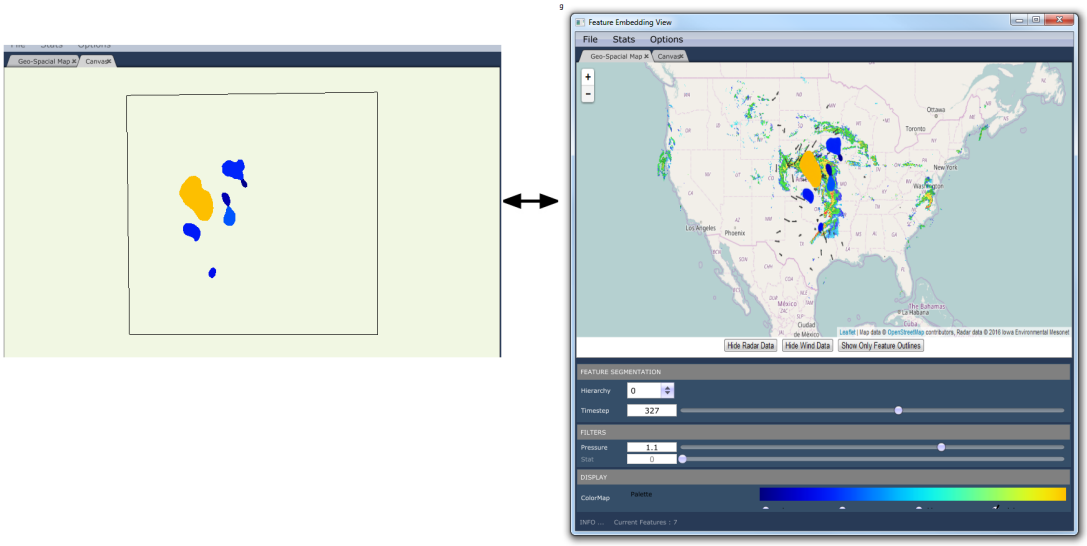


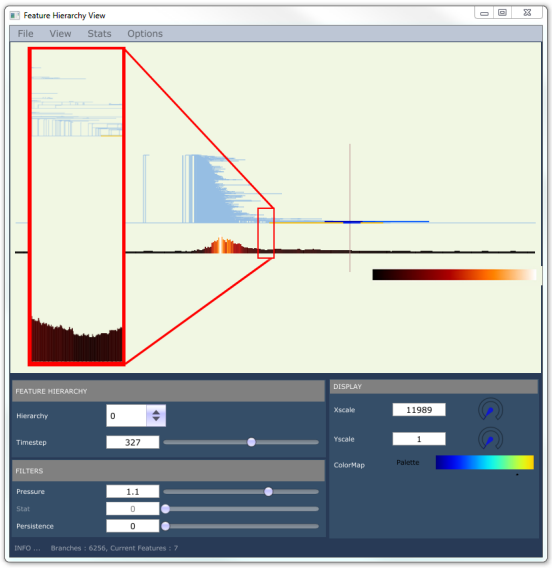
Figure 8.6: The tweets of organizations are suitable for exploring event progression that interests the public (right) whereas the tweets of individuals are good for day-to-day activities of individuals (left). For the highlighted feature tracks, the tweets that form some of its features are displayed.

understand atmospheric phenomena. Here, the proposed generic system is enhanced and extended to better fit the needs of atmospheric scientists, see Figure 8.7. Specifically, several new functionalities are added to the system such as simultaneous exploration of multiple feature hierarchies; histogram views that summarize the stability of features within each time step; and displaying each feature's geographic location and trajectory information within the geospatial view. Furthermore, to better understand the global atmospheric phenomena, pressure-perturbation events are visualized alongside other ancillary data sets such as radar imagery and wind observations. Figure 8.8 shows the data embedding view of the system displaying a feature's geographic location and trajectory information.

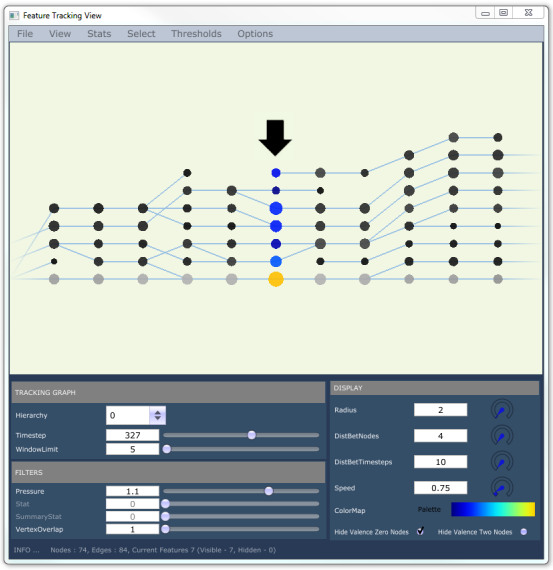
In addition to the atmospheric case studies presented in Sections 4.2.2, 5.2.1, 7.3.2, and 9.1, several other case studies have also been analyzed using the proposed generic framework and provided to atmospheric scientists for evaluation and feedback. According to the collaborating atmospheric scientists, the proposed generic system has been proven to be a powerful platform for conducting retrospective analysis to better understand different atmospheric phenomena. Specifically, the capability to explore features across multiple pressure-perturbation parameter values and the interactive abilities to thoroughly explore the tracking graphs are a distinct advantage over conventional techniques used in the field of atmospheric sciences. Also, the visualizations provided by the interactive platform al-



(a) Feature embedding view



(b) Feature hierarchy view



(c) Feature tracking view

Figure 8.7: The proposed system visualizing atmospheric data. (a) Shows the data embedding of features using two subcomponents: geometric embedding (left) and geospatial view (right). The geospatial view displays geographical information of features as well as various ancillary data such as radar and wind observations whereas the geometric embedding view displays the geometric information of the embedded features. (b) Visualizes the horizontal graph layout of the feature hierarchy along with a histogram-based view that summarizes the stability of features. Here, a zoomed-in view of the feature hierarchy (red box top) and histogram (red box bottom) is also displayed. Finally, (c) shows the evolution of features using a tracking graph for a user-selected focus time step (indicated with a black arrow) and a time window.

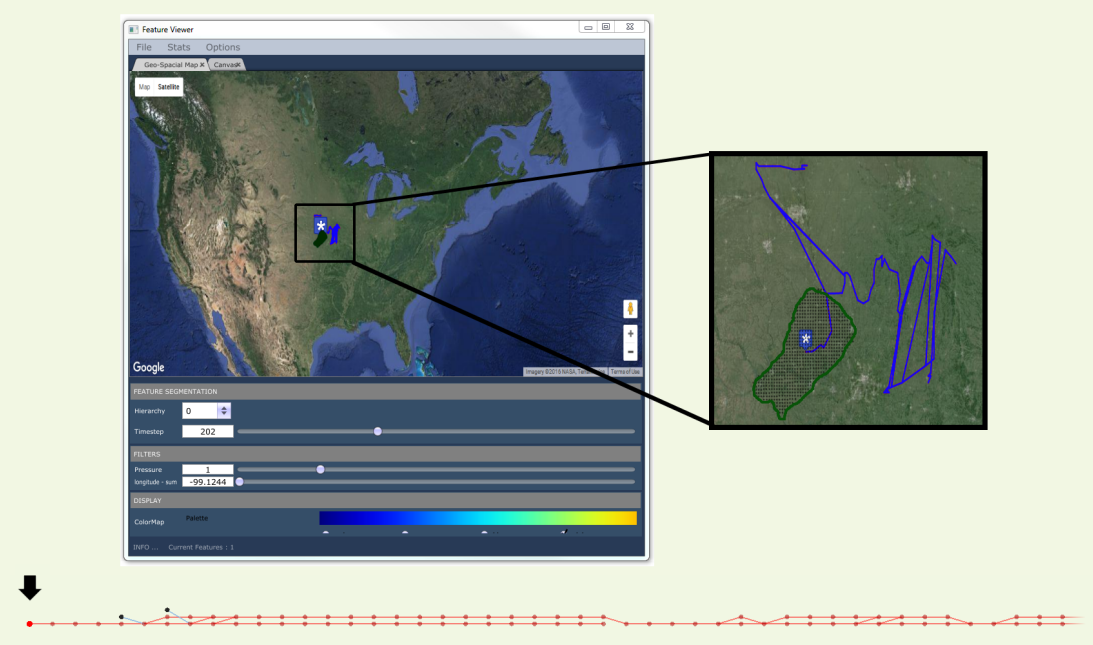


Figure 8.8: For atmospheric data, the geospatial view of the feature embedding view (top) showing a selected feature's geographic location, shape, and trajectory information. The feature shape is shown in green and the trajectory is in blue. The selected feature's track within the tracking graph is also displayed (bottom), and the focus time step within the graph is indicated with a black arrow.

low scientists to perform sensitivity analysis and to determine what pressure-perturbation parameter values appear valid for more accurate representations of various atmospheric phenomena. Furthermore, interactive assessment and filtering of feature properties, such as total feature area, feature lifetime, and feature movement (speed and direction), are also extremely useful for assessing which features been used for retrospective analysis of past atmospheric events. The collaborating scientists have indicated that it could also have potential use in 'nowcasting', which is short-term (0-6 hrs) forecasting of potential weather events. Specifically, the system's interactive capabilities to extract, filter, and study certain pressure-perturbation events are appreciated in this kind of real-time setting. More results related to this particular application example can be found in [176].

8.3 Summary

This chapter provides comprehensive details regarding a novel generic system designed to explore the evolution of features in dynamic data sets. Specifically, given the appropriate abstractions for feature definitions and their correspondences, this proposed

framework integrates efficient data structures and visualization techniques for addressing the general task of understanding feature evolution across time. This system also has leeway for including additional domain-specific functionality to better facilitate the requirements of scientists. As part of the research work that has been carried out during this dissertation, a variety of dynamic data sets have been used within this proposed visualization and analysis environment demonstrating its utility and applicability. Two real-word examples from social media and atmospheric science domains are presented in this chapter.

CHAPTER 9

COMPREHENSIVE CASE STUDIES IN FEATURE EVOLUTION EXPLORATION

This chapter presents several comprehensive case studies from the atmospheric science, combustion, and healthcare domains to demonstrate the applicability and generality of the approaches and techniques presented in this dissertation.

9.1 Evolution of Pressure-Perturbation Events in Atmospheric Data

Atmospheric sciences is the study of physical and chemical phenomena occurring in the Earth's atmosphere. This study entails understanding the state of the Earth's atmosphere, how it is changing across time and why. In particular, understanding how various weather events develop and evolve is often conducted through retrospective analysis of past atmospheric events. Atmospheric scientists can then utilize tools to better predict potential hazards and provide earlier warnings for events that may impact life and property. Several atmospheric state variables can be measured to identify high-impact events, one of which is surface atmospheric pressure. Many weather events can be characterized by variations in surface pressure from the mean pressure value (i.e., pressure-perturbations).

Accordingly, there is significant interest in extracting pressure-perturbations both spatially and temporally to better understand the evolution of weather events. The ability to interactively extract, explore, and characterize pressure-perturbations both in space and across time to better understand atmospheric phenomena has the potential to greatly improve the understanding of past weather events. Furthermore, it also provides forecasters the ability to 'nowcast' (i.e., a short-term forecast for 0-6 hrs) future events by assessing the pressure-perturbations with respect to other data sets such as radar imagery and wind observations on a single interactive interface.

Understanding these types of atmospheric phenomena requires identifying and exploring the feature of interest (i.e., pressure-perturbation event) for the entire data set. Typically, a pressure-perturbation event is defined to be a single region in the domain where all the pressure-perturbation values are above/below a given parameter value. Often, such events are referred to as positive or negative pressure-perturbation events, respectively. With the approaches and techniques presented in this dissertation, atmospheric scientists are able to extract pressure-perturbation events across multiple parameter values and explore their evolution with the aid of tracking graphs. Additionally, with the proposed visualization and analysis environment, scientists are allowed to interactively explore pressure-perturbation data sets. As discussed in Chapter 8, to better understand atmospheric phenomena, the proposed system provides support to leverage other conventional data sets such as radar imagery and wind observations.

In this section, a third case study extracted from the same pressure-perturbation data set as in Sections 4.2.2, 5.2.1, and 7.3.2 is considered. This pressure-perturbation data set contains data for the March 1 - August 31, 2011 period and is provided by [192]. The case study of interest contains discrete thunderstorm complexes, and is used to demonstrate the utility of the proposed system for exploring short-term pressure-perturbations with respect to other conventional atmospheric science data sets such as radar imagery and surface wind observations to assess high-impact phenomena such as thunderstorms.

In this case study, a first complex could be seen moving east across eastern Kansas into Missouri from 0300-0900 UTC June 18, while a second complex developed behind the first at 0600 UTC June 18 and moved east-northeast from 0600-1200 UTC June 18. Both thunderstorm complexes produced reports of wind damage, but the pressure-perturbations produced by these complexes varied in strength, highlighting the complexity of the physical processes associated with predicting and analyzing these events. Here, the most pronounced pressure features are the negative pressure-perturbation feature known as a “wake low” with the first complex, and the positive feature known as a “mesohigh” with the second complex [192]. The strength and proximity of these negative and positive features result in strong surface winds, which likely contributed to additional wind damage associated with the second complex. For this case study, Figure 9.1 displays radar imagery along with pressure-perturbations from the USArray TA stations courtesy of web

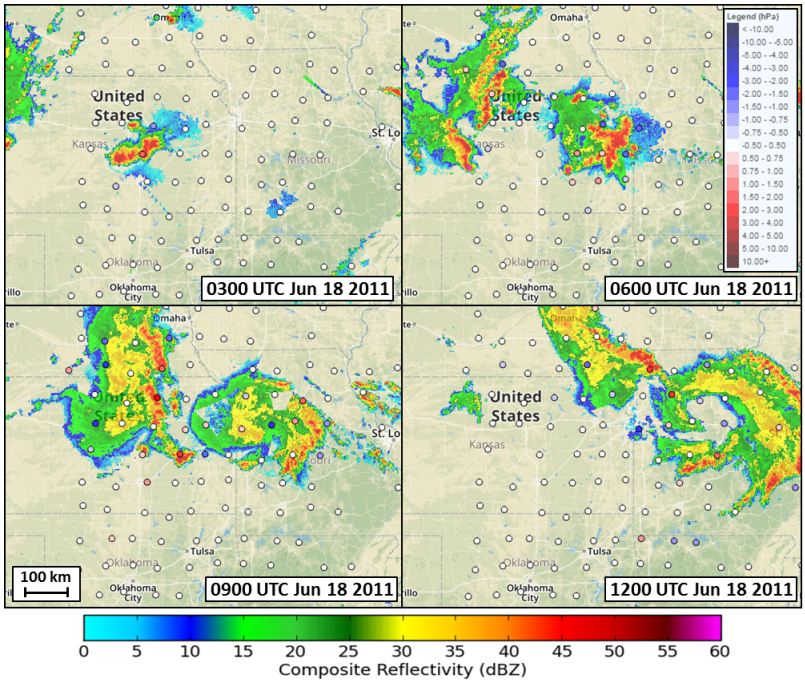


Figure 9.1: For the case study of interest, USArray mesoscale-filtered pressure-perturbations (hPa) are overlaid on composite radar reflectivity for 0300, 0600, 0900, and 1200 UTC June 18, 2011.

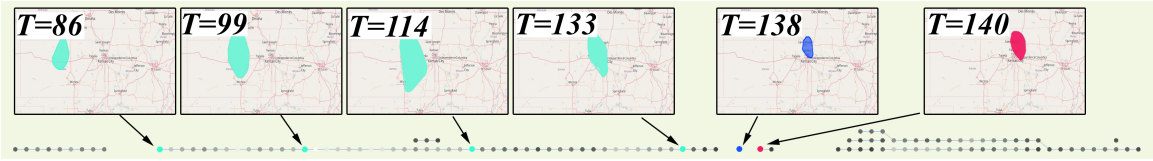
products [193] developed as a part of research by the University of Utah Department of Atmospheric Sciences.

The original data for this case study contain 192 time steps and total ≈350MB of raw data. First, an offline preprocessing step is used to compute the relevant feature hierarchies and the meta graphs. For this data set, scientists need to simultaneously explore both positive and negative pressure-perturbations events. They also wish to compare feature evolution results with respect to metrics based on spatial overlap and distance proximity. Therefore, for every time step of the case study, two feature hierarchies are computed to enable simultaneous exploration of positive and negative pressure-perturbation events, respectively. During construction, various feature-based attributes such as centroid, area, median magnitude value, maximum/minimum magnitude value, position of the maximum/minimum magnitude value, long-axis distance, short-axis distance, long-axis orientation, eccentricity, propagation distance, and propagation speed are also computed and stored in the feature hierarchies. Next, for exploring the feature evolution with respect to both feature correspondence metrics, the necessary feature correspondence details for

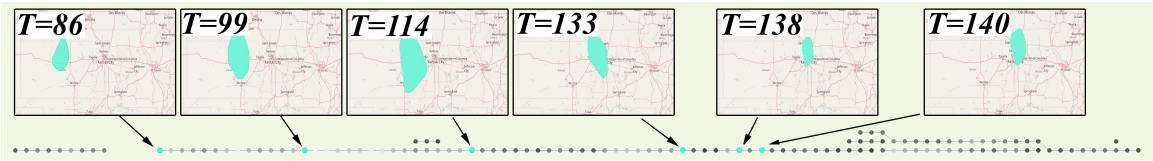
each metric are computed and stored in separate meta graph structures. Here, the feature correspondence amount (i.e., the amount of spatial overlap or the length of Euclidean distance between features) is stored as a feature correspondence-based attribute in each meta graph. Once the feature hierarchies and meta graphs are stored in the feature hierarchy and meta graph format, the total data size is reduced to around 10MB.

With respect to the research carried out as a part of this dissertation, several of its key aspects that assist in exploring this particular case study are described here. First, scientists are able to explore the evolution of the pressure-perturbation events with the use of tracking graphs. As shown in Figure 9.2a, the proposed system identifies the existing thunderstorms and visualizes their evolution across time.

Second, the tracking graph results provide insights into the underlying structure of the pressure-perturbation event. In this case, they indicate that the temporal evolution of one of the complexes (the one existing over eastern Kansas) is not very consistent across time. Further exploration reveals that this is due to the parameter choices and the complexity of this phenomenon. At the initial time steps, the thunderstorm complex is strong enough to be detected at the selected parameter values, but it loses its pressure magnitude across time. Here, the ability to define localized, per-feature parameter values in the proposed



(a) Tracking graph for 1.0 hPa



(b) Same tracking graph using adaptive parameter values

Figure 9.2: Evolution of the pressure-perturbation events visualized using tracking graphs. (a) A tracking graph showing the evolution of pressure-perturbation events for 113 time steps at 1.0 hPa. Here, one of the thunderstorm complexes (in green) is not consistent across time. This feature disappears at $T = 136$ and then reappears and dies at $T = 138$. Again, it reappears at $T = 140$ and later at $T = 146$. (b) The resulting tracking graph after per-feature, localized paramater values are used to locally modify the features (in green) to allow continuous feature tracking.

system is helpful in dynamically adapting the parameter choices within the tracking graph to allow for further detection of the pressure-perturbation event, see Figure 9.2b.

Next, with the use of the proposed framework, scientists are able to perform simultaneous exploration of multiple pressure-perturbation events across time. This capability is specifically included in the proposed framework to support atmospheric data sets. It enables scientists to simultaneously explore both positive and negative pressure-perturbation events and study their coupling structure, see Figure 9.3. In this case, the results provide insights into the strength of the coupling between the positive and negative pressure-perturbation events.

Finally, the ancillary information provided by the proposed framework is particularly useful in gaining new insights into the atmospheric phenomena. For a specific time step, scientists can compare pressure-perturbation events (at varying parameter values) with the relevant radar imagery and also explore the relationships of these events with wind observations. Figure 9.4 shows an instance where the strong coupling of positive- and negative-pressure-perturbation events appears to be collocated well with very strong recorded wind speeds, which is to be expected physically due to the high pressure-perturbation gradients. Upon further exploration it was found that these strong winds are not always perfectly collocated with radar observations, but rather they are located closer to the leading edge of the positive-pressure-perturbation event. This insight is of significant interest to scientists as conventional meteorological tools are not be able to detect and assess such a phenomenon. More results related to this particular application example can be found in [176].

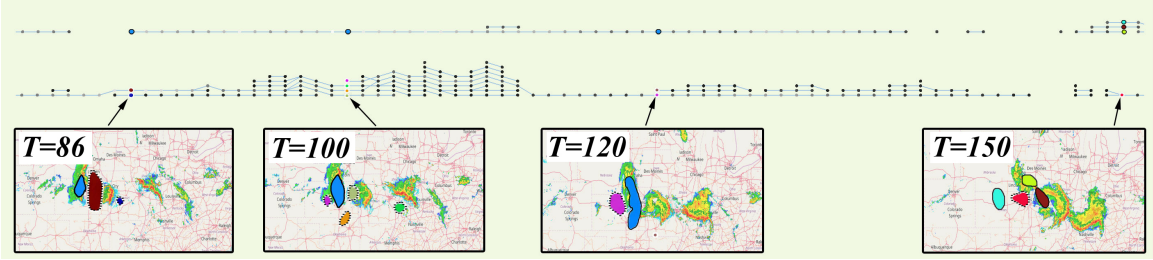


Figure 9.3: Simultaneously exploring both positive and negative pressure-perturbation events for multiple time steps. The graph on top shows the tracking graph at +1.0 hPa and the bottom at -1.0 hPa. Here, positive (outlined in solid black line) and negative (outlined in dashed black line) pressure-perturbation events existing at $T = 86, 100, 120$, and 150 time steps are also visualized.

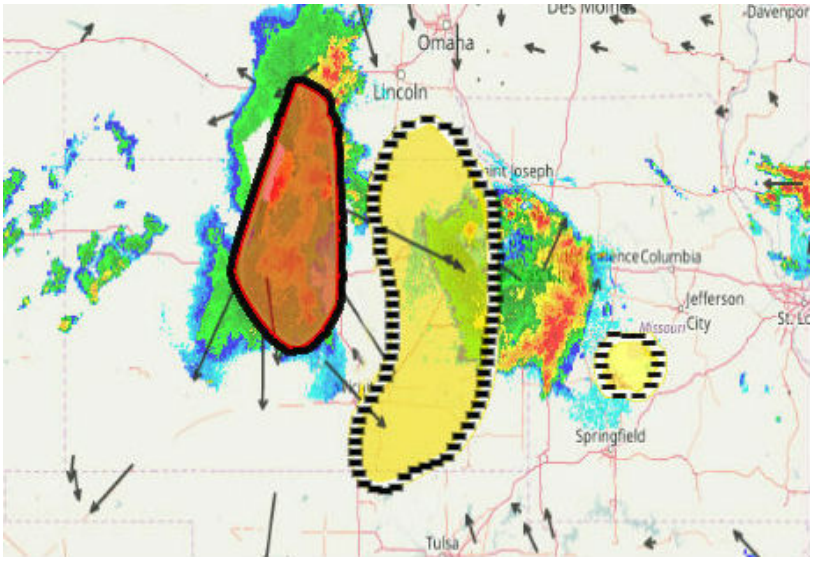
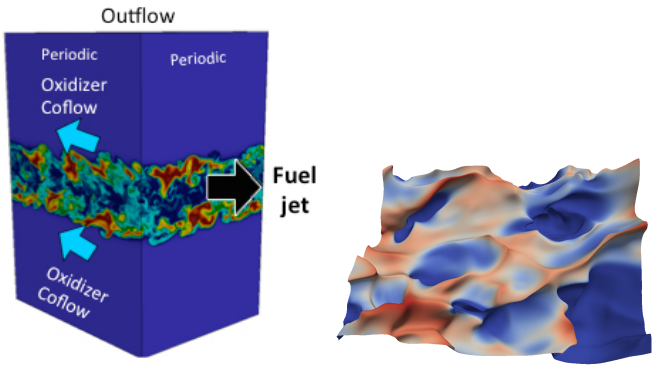


Figure 9.4: Both positive (outlined in solid black line) and negative (outlined in dashed black line) pressure-perturbation events are visualized alongside their radar imagery and wind observations (in black arrows). Here, the size and direction of winds are indicated by the arrow length and orientation. The image shows a strong coupling of positive- and negative- pressure-perturbation events. Such events are also collocated with strong winds going from the high- to low-pressure regions.

9.2 Evolution of Extinction Regions in Combustion Data

In this section, a large-scale, 3D direct numerical simulation (DNS) of a turbulent jet flame undergoing extinction and reignition is considered, see Figure 9.5. The data set contains 101 time steps at the resolution of $920 \times 1400 \times 720$, which even considering only the two scalar fields used here totals $\approx$1.5TB of data.

For this type of data, scientists define the flame as an isosurface of mixture fraction – the ratio of unburnt fuel versus combustion products – as shown in Figure 9.5b. However, not all this surface is considered burning. Instead, a second indicator, the OH concentration, is used to classify the flame into burning regions and extinction holes. A high OH level indicates an active chemical process, and a low concentration marks an extinct portion of the flame. As the simulation proceeds, more extinction holes develop, grow, and ultimately heal again and disappear. Tracking this process across time on a per-hole basis leads to new insights into why these holes are forming and what effects contribute to their closing. Therefore, scientists are very much interested in understanding the evolution of extinction regions in these turbulent flames.



(a) (b)

Figure 9.5: Details of the direct numerical simulation (DNS) of a turbulent jet flame: (a) Schematic of the DNS domain for the turbulent jet flame [194]. (b) The jet flame isosurface of mixture fraction with interpolated OH concentration. The blue regions represent extinction regions and red regions indicate an active chemical process. The isosurface is complex and can fold on itself, making feature tracking challenging.

However, a number of factors make this use case especially challenging for existing techniques. First, the exact OH parameter value to use is not known, and exploring how the evolution differs for different parameter values is of significant interest. However, given the size of the source data (≈1.5TB in total), a repeated analysis is practically infeasible, especially as part of an interactive exploration. Furthermore, most existing tracking techniques are based on spatial overlap or motion prediction [15], [125]. The former does not apply to these data as the flame surface itself moves quite significantly, and flame surfaces of successive time steps rarely overlap. Predictor-based approaches [17–19] work well for sparse sets of well-separated features in which ambiguities can be accurately resolved. Unfortunately, as shown in Figure 9.5b, the flame surfaces become highly convoluted, especially in later time steps, and features are quite densely present throughout the flame. This makes predictor-based schemes unreliable and difficult to tune. Weber et al. [195] propose to use a Reeb graph of a space-time surface to track such embedded features. However, this technique is quite involved, as well as expensive, and is applicable only to a fixed OH parameter value. Finally, there is a lack of temporal resolution in the original data set. Although this could be avoided through a repeated execution with more temporal resolution and/or in situ processing, re-running even parts of this specific simulation is infeasible due to time and space requirements.

For this particular data set, the approach proposed by Widanagamaachchi et al. [167] is

used to produce the equivalent feature tracking results of Weber et al. [195], but for variable parameter values. This algorithm builds an explicit space-time isovolume of mixture fraction between pairs of consecutive time steps and interpolates the corresponding OH values at all its vertices. Given a data set, an offline preprocessing step is used to compute the feature hierarchies and meta graph. Using the approach in [167], three feature hierarchies are computed for every two consecutive time steps. Next, feature correspondences are also computed in a preprocessing step to form the meta graph. This offline preprocessing step is performed in parallel at the Oak Ridge Leadership Computing Facility on Rhea a Linux cluster with 16, 2.0 GHx Intel Xeon processors per node. The resulting data, consisting of the feature hierarchies as well as the corresponding segmentation and tracking information, reduce the data size to about 6GB, the vast majority of which is dedicated to the spatial information for rendering.

Once the feature hierarchies and meta graph information are computed, the evolution of extinction regions can be explored for a range of OH parameter values. The resulting tracking graph shows the evolution of extinction regions assuming linear interpolation in time. However, depending on the temporal resolution of the original data, this graph may not be correct. For example, Figure 9.6 shows the evolution of extinction regions at the default OH concentration value of $9.77e^{-4}$, resulting in a complex, difficult to comprehend and process tracking graph, which is partially due to artifacts from the lack of temporal resolution and unavoidable instabilities in the parameter choices.

Consider the example shown in Figure 9.7. For a slow moving isosurface, as in t_0 to t_1, its interpolated OH field on the space-time isovolume displays the same structure as on the individual surfaces. Therefore, the two extinction holes, indicated by the blue sections on the surface, remain isolated and can be easily tracked through time. However, for a slightly faster moving isosurface, as in t_0 to t_2, its temporal interpolation creates artifacts. As shown in Figure 9.7c, the burning regions in red typically form pancake-like structures surrounding the flame surface. When the surface moves further than the width of the burning structures, the interpolation on the isovolume creates a single connected extinction region. This move results in the two extinction holes erroneously merging in the intermediate time step before splitting again. Such artificial merge-split events create artifacts in the tracking graph.

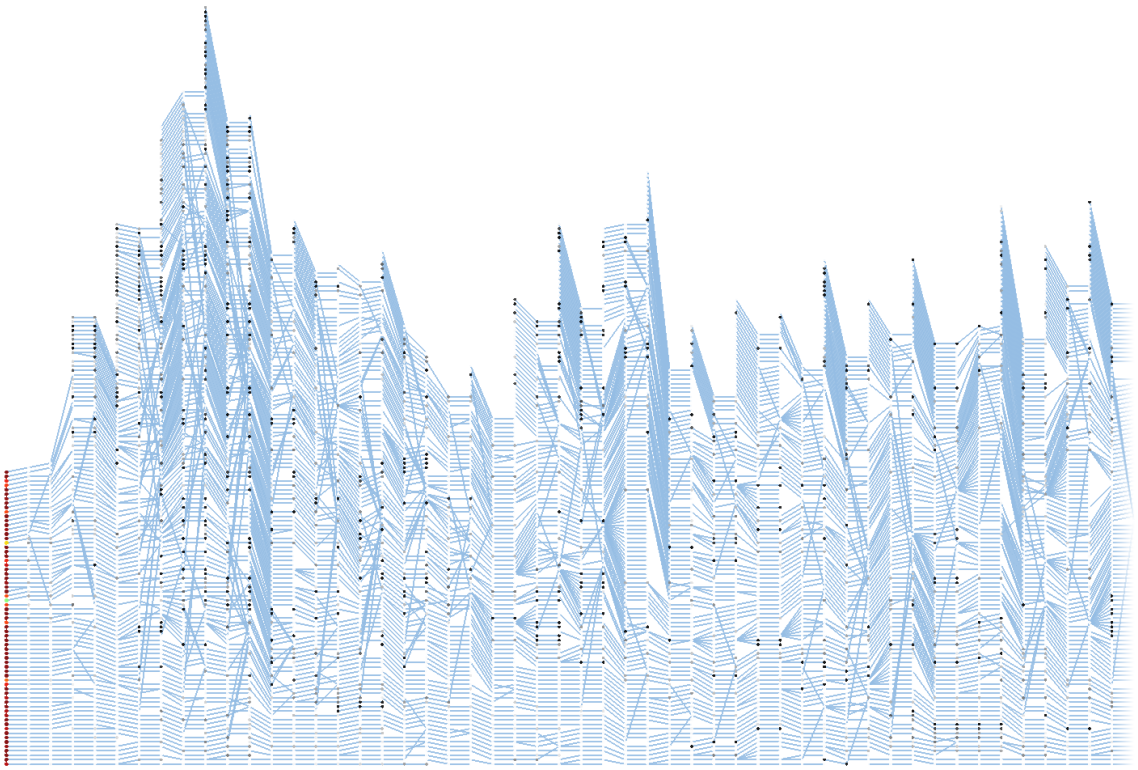
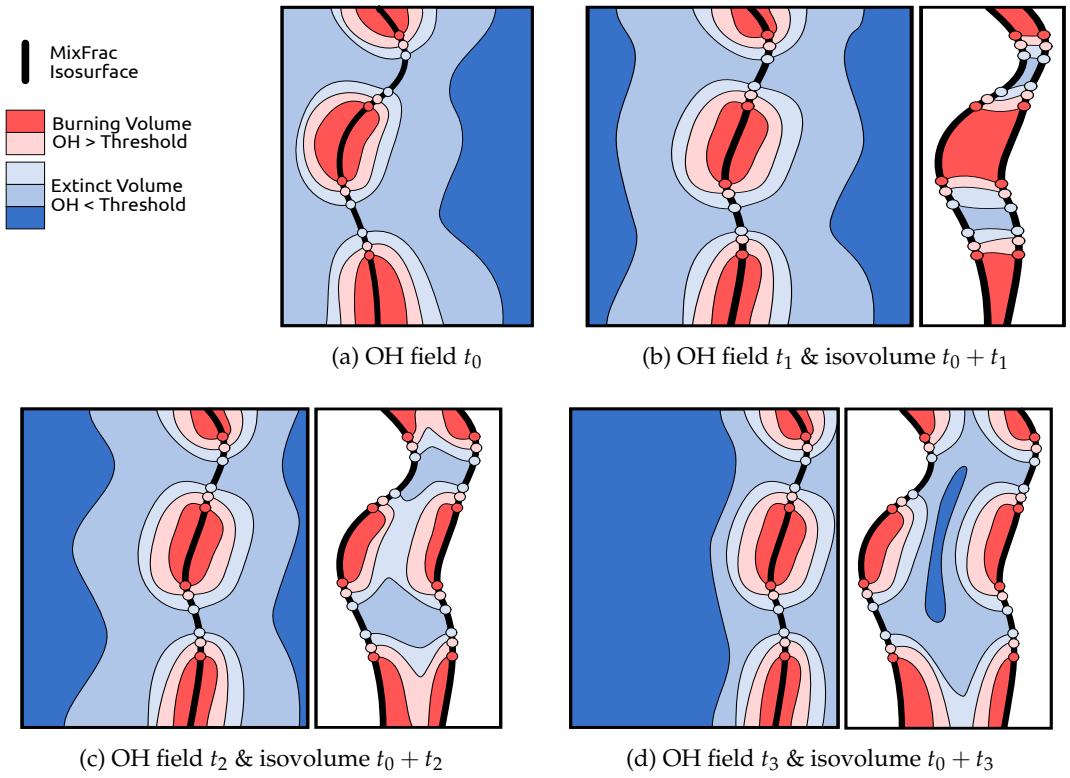


Figure 9.6: The tracking graph for the first 26 of the 101 time steps in the turbulent jet flame data set. As discussed in [167], an intermediate step between each consecutive pair is also displayed, so 51 total time steps are displayed. The graph contains only 1802 nodes, but still it is nearly incomprehensible due to the complex interactions.

These types of artifacts result from insufficient temporal resolution in the original data and can ideally be addressed by collecting frequent simulation snapshots. For this particular data set, it is too costly to repeat the simulation. Moreover, the temporal resolution available for postprocessing is strictly limited by the high cost of file I/O as well as storage space restrictions. Therefore, even if the simulation could be repeated, it is infeasible to save significantly more data. One option is to compute the tracking information in an in situ environment at the required temporal resolution. Unfortunately, this approach requires keeping at least two copies of the simulation state in memory, which is also not possible for this data set. Such processing also unduly increases the simulation time and file I/O, making it not viable at this point.

In such a setting, this dissertation attempts to alleviate some of these temporal artifacts by exploiting the flexibility to use localized parameter values. For the case in Figure 9.7c,



(a) OH field t_0

(b) OH field t_1 & isovolume $t_0 + t_1$

(c) OH field t_2 & isovolume $t_0 + t_2$

(d) OH field t_3 & isovolume $t_0 + t_3$

Figure 9.7: Effects of decreasing temporal resolution. (a) An illustration of the neighborhood around a mixture fraction isosurface. Small pockets of burning (high OH) fuel are separated by extinction holes (low OH). (b) A slow moving mixture fraction surface – t_0 to t_1 – creates easily tractable *tunnels* of low OH. (c) A moderately fast moving surface – t_0 to t_2 – artificially connects extinction holes. However, lowering the extinction parameter value to the middle shade of blue can compensate for this artifact. (d) A fast moving surface – t_0 to t_3 – can create a region of artificially low OH (dark blue) connecting extinction holes. These artifacts cannot be split by manipulating the parameter value.

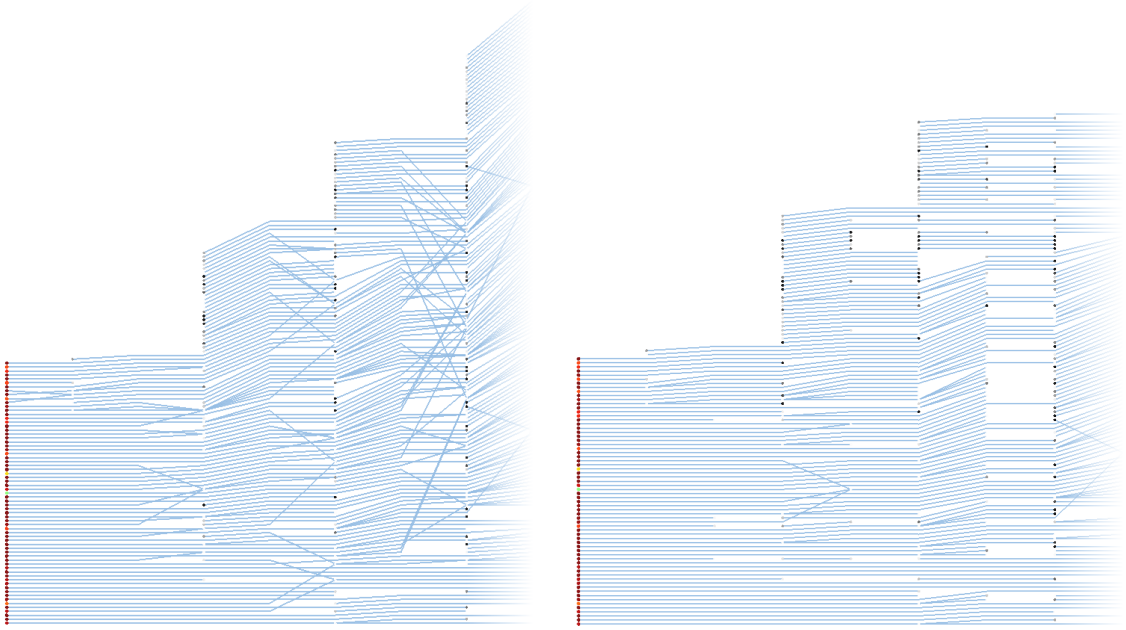
the interpolation connects otherwise separate extinction holes by a strip of relatively high, yet still extinct, OH values. This problem can be addressed by selectively lowering the OH parameter value of features, causing these artifacts in the intermediate time step. This change effectively reclassifies the light blue region as burning and separates the extinction holes across time to result in correct tracking information. This simple yet effective approach is able to alleviate many artifacts caused by the insufficient temporal resolution in data. However, some artifacts are too severe to be corrected in this manner. For example, when the isosurface is moving too fast, as shown in Figure 9.7d, a region of artificially low OH values, lower than those on the surface itself, is created in the isovolume. This artificial

connection cannot be cut by lowering the parameter value as the low valley in dark blue always exists.

When applying the proposed method, first, the tracking graph for a single parameter value is created. Then, all intermediate time steps of the tracking graph are searched for high valence, merge-split nodes indicating a potential temporal artifact. Next, the local parameter values of each merge-split node are progressively lowered until it either splits into individual nodes (i.e., resolving the artifacts) or until a further lowering increases the number of nonvalence two nodes in the tracking graph (i.e., disconnecting the extinction holes entirely). In each case, the parameter value change to the tracking graph is applied only in the first case (i.e., when lowering of the parameter value resolves the artifact and reduces the number of nonvalence two nodes in the tracking graph). It is important to note that use of localized parameter values does not introduce invalid features or correspondences. At any point, features used in the tracking are valid extinction regions even though they use slightly different parameter values. Moreover, as this proposed method selectively lowers the parameter value of features causing an artifact, only the tracking information in the intermediate time step is directly affected. Nevertheless, with the ground truth tracking information unavailable and the linear interpolation containing known artifacts, this approach remains a heuristic albeit a very effective one. Figure 9.8 shows an example of the first few time steps of a tracking graph for all extinction regions both before and after applying the proposed simplification method. More results related to this particular application example can be found in [167].

9.3 Progression of Patient Groups in Healthcare Data

As medical organizations increasingly adopt the use of electronic health records (EHRs), large volumes of clinical data are being captured on a daily basis. These data provide comprehensive information about patients and have the potential to improve a wide range of application domains in healthcare. Physicians and clinical researchers are interested in finding effective ways to understand this abundance of data. In this context, visualizing and exploring patient progression across time can provide valuable insights and facilitate the decision-making of physicians and clinical researchers.



(a) (b)

Figure 9.8: Reduction of temporal artifacts. (a) Early time steps of an original tracking graph for extinction holes. Here, a large number of artifacts exist, causing merge-split events in the intermediate time steps. (b) The same tracking graph after removing some of the temporal artifacts using localized parameter values.

However, several factors need to be taken into consideration when analyzing this type of data. First, given the large number of patients, an individual, per-patient analysis is time-consuming and does not lend itself to finding commonalities and trends. Instead, patients should be grouped according to various criteria, such as symptoms and outcomes. Second, to compare groups of patients who arrive at different times, their records must be aligned, for example, by their time of admission, time of major procedures, or other common factors. Third, patient progression across time needs to be presented in a concise manner to allow simultaneous exploration of large numbers of patients. Finally, the analysis must be interactive, allowing users to quickly explore different hypotheses within a hospital setting.

With the approaches and techniques presented in this dissertation, clinical researchers are allowed to interactively explore how patients and their progression change across time. They are able to extract patient groups across multiple patient similarities and study the progression of patients via interactive exploration of dynamically constructed tracking graphs.

In this section, the effectiveness of the approaches and techniques presented in this dissertation is demonstrated with the use of a publicly available intensive care unit (ICU) database. The clinical database of Multiparameter Intelligent Monitoring in Intensive Care (MIMIC II) databases [196] contains comprehensive EHR data collected from hospital medical information systems (both patient bedside workstations and hospital archives). This data set is obtained from a set of ICUs, including medical, surgical, coronary care, and neonatal, in a single tertiary teaching hospital in the 2001 to 2008 time period. It includes patient information that falls into various categories such as general, physiological, medications, fluid balance, notes, and reports, see Table 9.1. The entire database totals about $\approx 27GB$ and contains information about tens of thousands of ICU patients.

In order to visualize MIMIC II clinical data within the proposed visualization and analysis environment, the relevant patient hierarchies and meta graph structures need to be computed and stored. Again, the computation and storage are done in an offline preprocessing step. First, for each day in a patient's hospital stay, patient details available in the database (e.g., admission ID, age, gender, race, ICD-9 code, drug code, hospital stay length, mean heart rate, mean temperature, and maximum urine output) are extracted, which results in 38291 patient admissions from 32536 patients. These details are then aligned to make sure all admissions fall on the first time step of the resultant data set. The resulting data set, after aligning, contains 174 time steps (i.e., 174 days).

Patients in each time step are then clustered together using the metric of [197]. This

Table 9.1: An overview of the data categories in MIMIC II clinical database

General	Patient demographics, hospital admissions, discharge dates, room tracking, death dates (in or out of the hospital), ICD-9 codes, unique code for healthcare provider, and type (RN, MD, RT, etc).
Physiological	Hourly vital sign metrics, SAPS, SOFA, ventilator settings, etc.
Medications	IV meds, provider order entry data, etc.
Lab Tests	Chemistry, hematology, ABGs, imaging, etc.
Fluid Balance	Intake (solutions, blood, etc), output (urine, estimated blood loss, etc).
Notes & Reports	Discharge summary, nursing progress notes, etc; cardiac catheterization, ECG, radiology, and echo reports.

patient similarity metric was previously applied to the same MIMIC II clinical database to identify patient similarities on the first day of the ICU stay [197]. When applying this metric, the required clinical, administrative, and categorical variables are extracted from the database for each day of a patient's hospital stay. Next, correspondences across patient groups are computed by tracking individual patients within the groups. Once the patient groups and their correspondence details are computed and stored, the total data size is reduced to $\approx 680MB$.

As the necessary information is precomputed and stored using efficient data structures, researchers are provided with the flexibility to vary the patient similarity values and explore the entire parameter space interactively. Such interaction provides an understanding of how patients group together for varying similarity values during a particular day in their hospital stay. For example, Figure 9.9 shows several examples of patient groups and their progression for 30, 34, 35, 36, and 38 similarity values. As the similarity values decrease, more patients are grouped together, reducing the complexity of the tracking graph. Here, for a specific similarity metric, exploring the full range of similarities enables researchers to gain insights into that metric's range of values. For the patient similarity metric of [197], its appropriate similarity value range for this particular data set is 30-38. Any similarity value below or above that range either grouped all patients into one group or divided each patient to be in a separate group.

The full tracking graph showing the patient progression across time at 36 similarity value is displayed in Figure 9.10. By observing these tracking graphs, specifically feature track length indicating the hospital length of stay of patients, it is clear that although many of the patient stays are less than 90 days (i.e., 3 months), this data set also contains several longer patient stays. Specifically, of 32536 patients, 6 patients with hospital lengths of stay greater than 90 days are found.

More importantly, various simplification options available in the proposed system allow researchers to further simplify these tracking graphs. For instance, filtering the tracking graph by correspondence amount allows removal of the least frequent patient progression paths from the tracking graph, making frequent patterns more prominent. If an analysis is to be conducted only on longer hospital stays of patients, filtering options available in the proposed system, specifically filtering tracking graphs by the length of a

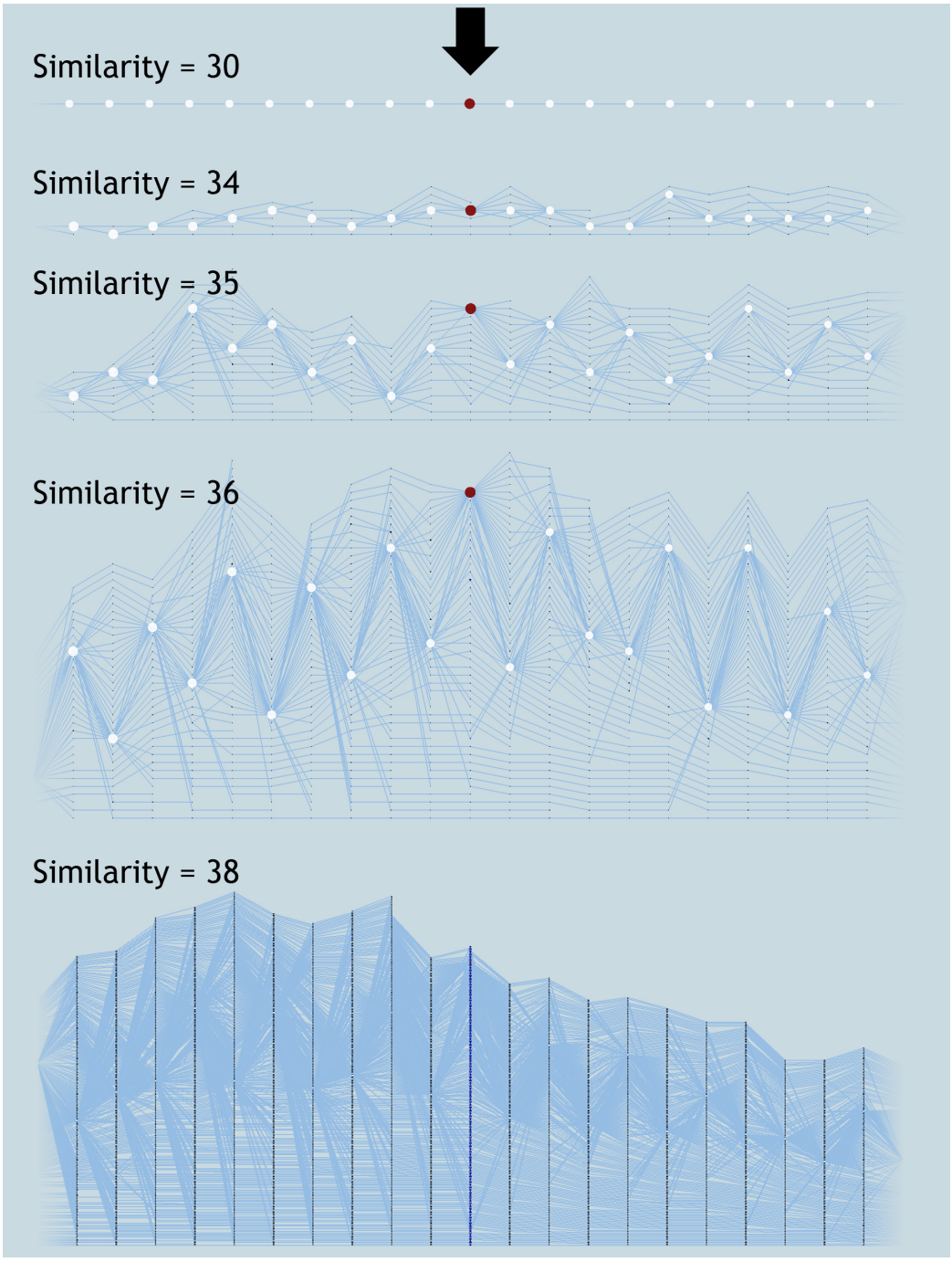


Figure 9.9: Effects of varying the similarity value to explore the temporal progression of patients. Here, patient groups and a portion of their corresponding tracking graphs are shown at 30, 34, 35, 36, and 38 similarity values. The focus time step of the tracking graphs is indicated with a black arrow, and the nodes are scaled based on the patient group's size. In each graph, patient progression for 10 time steps both forward and backward in time from the focus time step is displayed.

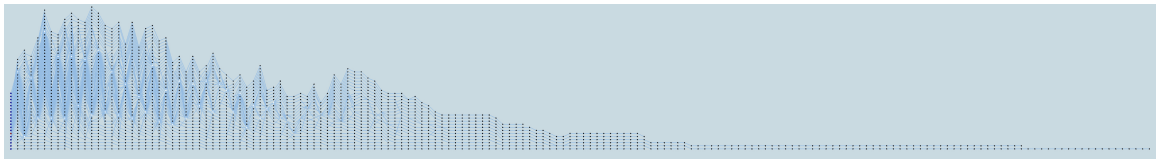


Figure 9.10: The entire tracking graph showing the complete patient progression for the MIMIC II clinical database. The graph contains 1110 nodes and 1288 edges for a total of 174 time steps. Here, the 36 similarity value is used.

feature track, are useful. Figure 9.11 illustrates several graph simplification results.

Additionally, as discussed in Chapter 8, the data embedding view of the proposed framework is useful for obtaining an overview of patient groups. As the user selects a certain patient group, this view displays the details of its patients. As shown in Figure 8.1(d), the word cloud visualization provides a quick visual overview of the information for a selected patient group. The numerical attributes such as age and hospital length of stay are converted to ranges to obtain more intuitive results. As shown in Figure 8.1(e), the exact patient details are also presented in the textual visualization within the system. More results related to this particular application example can be found in [198], [199].

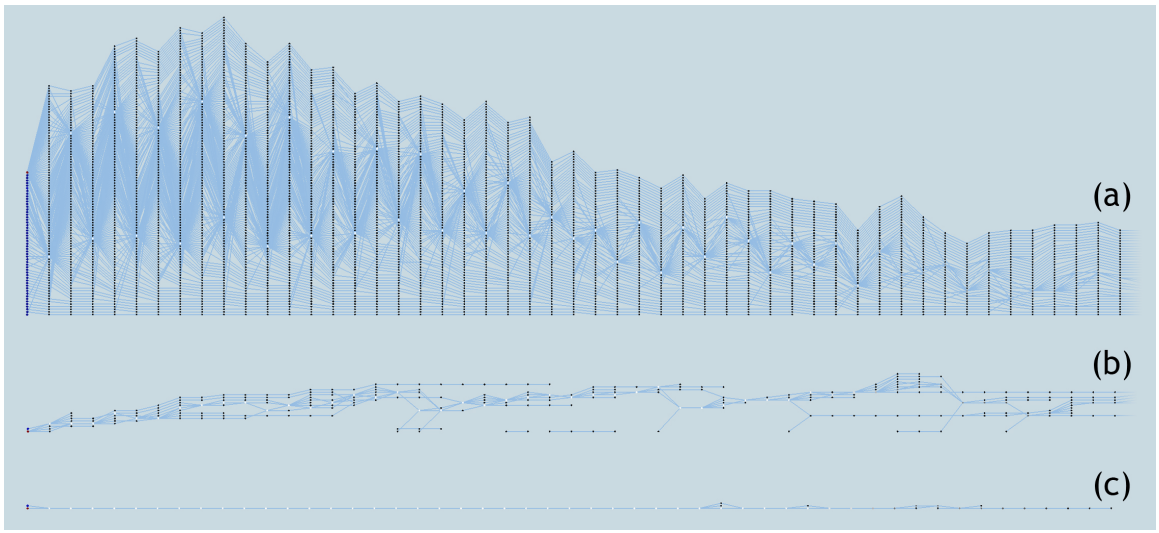


Figure 9.11: Tracking graph simplification. (a) A tracking graph showing the patient progression for the first 50 days within the hospital stay at the 37 similarity value. (b) Tracking graph in (a) filtered to contain only correspondences with overlap$\geq$2. (c) Tracking graph in (a) filtered to contain only patient groups with size$\geq$5.

9.4 Summary

This chapter provides comprehensive details on several case studies to demonstrate the utility and generality of the approaches and techniques presented in this dissertation. In particular, case studies from the combustion, atmospheric science, and healthcare domains are discussed. The research carried out as a part of this dissertation provides great flexibility in exploring the evolution of features across time for understanding the dynamic nature of each case study. First, the proposed multiresolution unified spatiotemporal representation of features is able to efficiently encode features and their correspondences for a range of parameter values. Second, the proposed layout and visualization strategies, together with the proposed visualization and analysis framework, provide the flexibility to interactively explore the feature evolution for a range of parameter values. In this manner, by precomputing the feature hierarchies and meta graph structure and storing them using optimized data structures, and then utilizing efficient visualization techniques, this dissertation allows interactive exploration of features across time for several gigabytes of data. Furthermore, due to the generality of their design, as demonstrated in this chapter, the proposed approaches and techniques of this dissertation are applicable across a wide variety of application domains.

PART V

CONCLUSION AND FUTURE WORK

CHAPTER 10

SUMMARY AND OUTLOOK

Here, a summary and outlook on the work described in this dissertation is presented.

10.1 Summary

The increase in our ability to gather and generate data enables a wide range of application domains to routinely capture dynamic data at an unprecedented scale. Irrespective of the application domain, one of the most common analysis tasks is the need to understand the dynamic aspects of these increasingly large data sets. A dynamic data set often contains some notion of a feature of interest at each moment in time, and these features evolve across time. As a result, exploring and analyzing the spatiotemporal behaviors of features is considered to be one intuitive way of understanding the dynamic aspects of these data sets. Although numerous advances have been made on the topic of feature evolution exploration within dynamic data sets, many important challenges still remain for several reasons, including the resolution and complexity of dynamic data sets, physical limitations of human visual system, and limitations of the existing approaches.

To address some of these challenges, this dissertation presents a non-domain-specific approach for exploring feature evolution across time. First, this dissertation introduces a novel multiresolution unified spatiotemporal representation of features that enables interactive extraction of feature evolution across time (Part II, Chapters 3, 4, 5). This representation provides the means to efficiently encode features and their correspondences for a range of parameter values, which can then be used to quickly and easily extract feature evolution details for any particular parameter value within the entire parameter range. Second, with a focus on tracking graphs, a novel progressive layout and visualization strategy that enables interactive visualization of feature evolution details is introduced (Part III, Chapter 6). By processing a tracking graph with respect to a focus time step and a window of interest, and then utilizing a progressive two-stage graph layout algorithm, this

proposed strategy provides a flexible approach to visualize and explore feature evolution across time. Third, to better understand the underlying patterns and trends in dynamic data sets, several progressive strategies that reduce the visual complexity of feature evolution details are introduced (Part III, Chapter 7). In particular, two graph subselection-based approaches, filtering and feature selecting, are presented for extracting subgraphs from a tracking graph and another localized, per-feature parameter value-based approach that exploits the flexibility in defining temporally and spatially varying parameters for simplifying tracking graphs is also presented. Fourth, a novel visualization and analysis environment that couples the multiresolution unified spatiotemporal representation of features with the progressive layout and visualization strategy for understanding feature evolution across time in a more general fashion is presented (Part IV, Chapter 8). As this framework focuses on interactively exploring and analyzing feature evolution in a non-domain specific manner, the necessary generalization within this system is maintained through abstraction, and the utility and generality of this proposed framework are demonstrated by using dynamic data sets from a range of application domains. In summary, the research done as part of this dissertation indicates that given the appropriate abstraction for feature definitions and their correspondences, the challenge reverts to finding good visual metaphors independent of the data type or the community providing the solution, which then leads to more general approaches instead of more specialized ones.

10.2 Directions for Future Research

The work described in this dissertation, in its current form, is useful for a variety of application domains, yet it can be further improved in a number of ways to make it even more widely applicable. First of all, the current work is limited to univariate and bivariate features. The work presented in Part II introduces a multiresolution unified spatiotemporal representation of features that is limited to univariate and bivariate features. As a result, the proposed visualization and analysis environment for exploring feature evolution over time is also limited to univariate and bivariate features. Nevertheless, as a concept, the same idea can be extended to multivariate features as well. Specifically, the proposed hierarchical feature representations can be extended to multivariate features by following the same approach as in the bivariate case – starting from the individual univariate feature

hierarchies, those hierarchies can be combined two at a time in a bottom-up fashion to produce a multivariate feature hierarchy.

Second, each component of the proposed multiresolution unified spatiotemporal representation of features can be further improved or extended. For instance, instead of being limited to only disjoint nested feature hierarchies, the proposed bivariate graph can be extended to handle other types of feature hierarchies. Here, as it involves handling the overlapping property, constructing a bivariate graph by combining two overlapping feature hierarchies will be a particularly interesting research problem. Furthermore, the meta-graph structure can be improved such that instead of encoding only the feature correspondences across features whose lifetimes overlap, all possible feature correspondences can be computed and stored using an efficient file format. At run time, a progressive strategy can be utilized to read in the feature correspondences as necessary. Initially, only feature correspondences across features whose lifetimes overlap can be read; other feature correspondences can be read as necessary.

Another research direction is exploring alternative file formats for the proposed data representations. By preserving locality for fast spatiotemporal query searches, efficient file formats can be designed for storing both features and their correspondence details. For a specific spatiotemporal query, such an approach allows portions of the files to be read, and then those portions of data can be used to compute the required feature evolution details. These aforementioned research directions require further investigation to efficiently encode features and correspondences for the entire parameter range and to support the necessary abstraction to maintain the generality.

Finally, numerous improvements can be made on the tracking graphs. Instead of the progressive two-stage graph layout algorithm, alternative graph layout strategies that can preserve the interactivity for larger tracking graphs and yet produce optimized layouts can be explored. Also, when scaling to larger data sets, techniques that reduce the visual clutter in tracking graphs (e.g., edge bundling) and other graph summarization approaches can be employed to better convey the underlying trends in data. To that end, identifying the salient time steps of a data set and constructing a summarized global tracking graph is another interesting research direction.

REFERENCES

[1] K. Heitmann, N. Frontiere, C. Sewell, S. Habib, A. Pope, H. Finkel, S. Rizzi, J. Insley, and S. Bhattacharya, "The q continuum simulation: Harnessing the power of gpu accelerated supercomputers," *Astrophys. J. Suppl. Series*, vol. 219, no. 2, p. 34, 2015.

[2] J. Brandon. (2015) How to collect and analyze data from 100,000 weather stations. [Online]. Available: http://www.cio.com/article/2936592/big-data/how-to-collect-and-analyze-data-from-100000-weather-stations.html

[3] P. Nyberg. (2014) 3 ways big data, supercomputing change weather forecasting. [Online]. Available: http://www.informationweek.com/big-data/big-data-analytics/3-ways-big-data-supercomputing-change-weather-forecasting/a/d-id/1269439

[4] H. Chang, "Book review: Data-driven healthcare and analytics in a big data world," *Healthcare Informatics Res.*, vol. 21, no. 1, pp. 61–62, 2015.

[5] J. Desjardins. (2016) What happens in an internet minute in 2016? [Online]. Available: http://www.visualcapitalist.com/what-happens-internet-minute-2016

[6] J. van Oijen, A. Donini, R. Bastiaans, J. ten Thije Boonkkamp, and L. de Goey, "State-of-the-art in premixed combustion modeling using flamelet generated manifolds," *Progress in Energy and Combustion Science*, vol. 57, pp. 30–74, 2016.

[7] J. M. McGinnis, L. Olsen, W. A. Goolsby, C. Grossmann *et al.*, *Clinical Data as the Basic Staple of Health Learning: Creating and Protecting a Public Good: Workshop Summary*. National Academies Press, 2011.

[8] I. L. Stats. (2017) Twitter usage statistics. [Online]. Available: http://www.internetlivestats.com/twitter-statistics/

[9] D. Kowalczyk, "Nonscientific and scientific research: Definitions and differences," 2015, [Online; accessed 10-March-2017]. [Online]. Available: http://study.com/academy/lesson/nonscientific-and-scientific-research-definitions-and-differences.html#transcriptHeader

[10] P. I. Lazorko, "Science and non-science," 2017, [Online; accessed 10-March-2017]. [Online]. Available: https://philosophynow.org/issues/96/Science_and_Non-Science

[11] T. Sørlie, R. Tibshirani, J. Parker, T. Hastie, J. Marron, A. Nobel, S. Deng, H. Johnsen, R. Pesich, S. Geisler *et al.*, "Repeated observation of breast tumor subtypes in independent gene expression data sets," *Proc. National Academy of Sciences*, vol. 100, no. 14, pp. 8418–8423, 2003.

[12] R. Butterworth, G. Piatetsky-Shapiro, and D. A. Simovici, "On feature selection through clustering," in *Proc. 5th IEEE Int. Conf. Data Mining*. IEEE, 2005, pp. 4–pp.

[13] B.-S. Sohn and C. Bajaj, "Time-varying contour topology," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 1, pp. 14–25, 2006.

[14] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell, "Interactive exploration and analysis of large-scale simulations using topology-based data segmentation," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 9, pp. 1307–1324, 2011.

[15] D. Silver and X. Wang, "Tracking and visualizing turbulent 3d features," *IEEE Trans. Vis. Comput. Graphics*, vol. 3, no. 2, pp. 129 –141, apr-jun 1997.

[16] ——, "Tracking scalar features in unstructured data sets," in *Proc. Conf. Visualization'98*. IEEE, 1998, pp. 79–86.

[17] R. Samtaney, D. Silver, N. Zabusky, and J. Cao, "Visualizing features and tracking their evolution," *Comput. J.*, vol. 27, no. 7, pp. 20–27, July 1994.

[18] F. Reinders, F. H. Post, and H. J. W. Spoelder, "Visualization of time-dependent data using feature tracking and event detection," *Int. Symp. Visual Comput.*, vol. 17, pp. 55–71, 2001.

[19] F.-Y. Tzeng and K.-L. Ma, "Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations," in *Proc. 2005 ACM/IEEE Conf. Supercomputing (SC)*. IEEE Computer Society, 2005, p. 6.

[20] C. Muelder and K.-L. Ma, "Rapid feature extraction and tracking through region morphing," Citeseer, Tech. Rep., 2007.

[21] W.-H. Hsu, J. Mei, C. D. Correa, and K.-L. Ma, "Depicting time evolving flow with illustrative visualization techniques," in *ArtsIT*, 2009, pp. 136–147.

[22] A. Joshi, J. Caban, P. Rheingans, and L. Sparling, "Case study on visualizing hurricanes using illustration-inspired techniques," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 5, pp. 709–718, 2009.

[23] J. Wilhelms and A. V. Gelder, "Octrees for faster isosurface generation," *ACM Trans. Graph.*, vol. 11, no. 3, pp. 201–227, Jul. 1992.

[24] H.-W. Shen, L.-J. Chiang, and K.-L. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree," in *Proc. Conf. Visualization '99: Celebrating Ten Years*. IEEE Computer Society Press, 1999, pp. 371–377.

[25] E. B. Lum, K. L. Ma, and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," in *Proc. Conf. Visualization'01*, ser. VIS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 263–270. [Online]. Available: http://dl.acm.org/citation.cfm?id=601671.601712

[26] K.-L. Ma, A. Stompel, J. Bielak, O. Ghattas, and E. J. Kim, "Visualizing very large-scale earthquake simulations," in *Proc. Conf. Supercomputing (SC)*. ACM, 2003, p. 48.

[27] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. Gao, H. Qu, and X. Tong, "Textflow: Towards better understanding of evolving topics in text," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2412–2421, 2011.

[28] W. Dou, L. Yu, X. Wang, Z. Ma, and W. Ribarsky, "Hierarchicaltopics: Visually exploring large text collections using topic hierarchies," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2002–2011, 2013.

[29] S. Liu, Y. Wu, E. Wei, M. Liu, and Y. Liu, "Storyflow: Tracking the evolution of stories," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2436–2445, Dec. 2013.

[30] P. Xu, Y. Wu, E. Wei, T.-Q. Peng, S. Liu, J. J. H. Zhu, and H. Qu, "Visual analysis of topic competition on social media," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2012–2021, 2013.

[31] C. Taylor. (2016) What are time series graphs? [Online]. Available: http://statistics.about.com/od/Descriptive-Statistics/a/Time-Series-Graphs.htm

[32] F. Beck, M. Burch, C. Vehlow, S. Diehl, and D. Weiskopf, "Rapid serial visual presentation in dynamic graph visualization," in *2012 IEEE Symp. on Visual Languages and Human-Centric Comput. (VL/HCC)*. IEEE, 2012, pp. 185–192.

[33] B. Bedat and R. K. Cheng, "Experimental study of premixed flames in intense isotropic turbulence," *Proc. Combust. Flame*, vol. 100, pp. 485–494, 1995.

[34] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell, "Analyzing and tracking burning structures in lean premixed hydrogen flames," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 2, pp. 248–260, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2009.69

[35] E. Koutsofios and S. North, "Drawing graphs with dot," AT&T Bell Laboratories, Murray Hill, NJ, Tech. Rep. 910904-59113-08TM, 1991.

[36] IBM. (2011) What is big data? [Online]. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[37] R. Pant. (2015) Visual marketing: A pictures worth 60,000 words. [Online]. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[38] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM Siggraph Comput. Graph.*, vol. 21, no. 4. ACM, 1987, pp. 163–169.

[39] I. Fujishiro, Y. Maeda, and H. Sato, "Interval volume: a solid fitting technique for volumetric data display and analysis," in *Proc. 6th Conf. Visualization'95*. IEEE Computer Society, 1995, p. 151.

[40] R. Adams and L. Bischof, "Seeded region growing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 6, pp. 641–647, 1994.

[41] T. Van Walsum, F. H. Post, D. Silver, and F. J. Post, "Feature extraction and iconic visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 2, no. 2, pp. 111–119, 1996.

[42] S. Havre, E. Hetzler, P. Whitney, and L. Nowell, "Themeriver: Visualizing thematic changes in large document collections," *IEEE Trans. Vis. Comput. Graphics*, vol. 8, no. 1, pp. 9–20, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1109/2945.981848

[43] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman, "Temporal event sequence simplification," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2227–2236, 2013.

[44] K. Wongsuphasawat and D. Gotz, "Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization." *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2659–2668, 2012.

[45] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *J. Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[46] V. Kumar and S. Minz, "Feature selection," *SmartCR*, vol. 4, no. 3, pp. 211–229, 2014.

[47] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *Science and Information Conf. (SAI), 2014*. IEEE, 2014, pp. 372–378.

[48] N. Elavarasan and D. K. Mani, "A survey on feature extraction techniques," *Int. J. Innovative Res. in Comput. and Commun. Eng.*, vol. 3, no. 1, 2015.

[49] H. A. Elnemr, N. M. Zayed, and M. A. Fakhreldein, "Feature extraction techniques: Fundamental concepts and survey," *Handbook of Research on Emerging Perspective in Intelligent Pattern Recog. Anal. and Image Process.*, 2015.

[50] W. Pin, Y. Hongjie, S. Weilie, Z. Yonghua, and G. Honghao, "Research on feature extraction based on deep learning," *Int. J. Hybrid Inform. Technol.*, vol. 8, no. 11, pp. 113–120, 2015.

[51] Wikipedia, "Hierarchy," 2003, [Online; accessed 11-March-2017]. [Online]. Available: https://en.wikipedia.org/wiki/Hierarchy

[52] M. Morse, "Relations between the critical points of a real function of independent variables," *Trans. Amer. Math. Soc.*, vol. 27, no. 3, pp. 345–396, 1925.

[53] J. W. Milnor, *Morse theory*. Princeton University Press, 1963, no. 51.

[54] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust on-line computation of reeb graphs: Simplicity and speed," in *ACM Tran. on Graph. (TOG)*, vol. 26, no. 3. ACM, 2007, p. 58.

[55] H. Doraiswamy and V. Natarajan, "Efficient algorithms for computing reeb graphs," *Proc. Annu. Symp. Computational Geometry*, vol. 42, no. 6, pp. 606–616, 2009.

[56] R. L. Boyell and H. Ruston, "Hybrid techniques for real-time radar simulation," in *Proc. November 12-14, 1963, Fall Joint Comput. Conf.* ACM, 1963, pp. 445–458.

[57] H. Freeman and S. Morse, "On searching a contour map for a given terrain elevation profile," *J. Franklin Inst.*, vol. 284, no. 1, pp. 1–25, 1967.

[58] M. Van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore, "Contour trees and small seed sets for isosurface traversal," in *Proc. 13th Annu. Symp. Computational Geometry*. ACM, 1997, pp. 212–220.

[59] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," *Proc. Annu. Symp. Computational Geometry*, vol. 24, no. 2, pp. 75–94, 2003.

[60] H. Carr, "Topological manipulation of isosurfaces," Ph.D. dissertation, Citeseer, 2004.

[61] V. Pascucci and K. Cole-McLaughlin, "Parallel computation of the topology of level sets," *J. Algorithmica*, vol. 38, no. 1, pp. 249–268, 2004.

[62] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. H. Chen, "Topological feature extraction for comparison of terascale combustion simulation data," in *Topological Methods in Data Analysis and Visualization*. Springer, 2011, pp. 229–240.

[63] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 5, pp. 1053–1060, 2006.

[64] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann, "Efficient computation of morse-smale complexes for three-dimensional scalar functions," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 6, pp. 1440–1447, 2007.

[65] S. D. White and C. S. Frenk, "Galaxy formation through hierarchical clustering," *Astrophys. J.*, vol. 379, pp. 52–79, 1991.

[66] M. Iwayama and T. Tokunaga, "Hierarchical bayesian clustering for automatic text classification," in *Proc. 14th Int. Joint Conf. Artificial Intelligence-Volume 2*. Morgan Kaufmann Publishers Inc., 1995, pp. 1322–1327.

[67] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets," in *Proc. 11th Int. Conf. Information and Knowledge Management*. ACM, 2002, pp. 515–524.

[68] S. Bandyopadhyay and E. J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks," in *INFOCOM 2003. 22nd Annu. Joint Conf. IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 1713–1723.

[69] D. Cai, X. He, Z. Li, W.-Y. Ma, and J.-R. Wen, "Hierarchical clustering of www image search results using visual, textual and link information," in *Proc. 12th Annu. ACM Int. Conf. on Multimedia*. ACM, 2004, pp. 952–959.

[70] J. A. Hartigan and J. Hartigan, *Clustering Algorithms*. Wiley New York, 1975, vol. 209.

[71] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *Comput. J.*, vol. 26, no. 4, pp. 354–359, 1983.

[72] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *J. Classification*, vol. 1, no. 1, pp. 7–24, 1984.

[73] C. F. Olson, "Parallel algorithms for hierarchical clustering," *J. Parallel Comput.*, vol. 21, no. 8, pp. 1313–1325, 1995.

[74] D. M. Blei, "Hierarchical clustering," *Lecture Slides, February*, 2008.

[75] G. Stanley and P. Reynolds, "Similarity grouping of australian universities," *Higher Education*, vol. 27, no. 3, pp. 359–366, 1994.

[76] N. Izadpanah, "A divisive hierarchical clustering-based method for indexing image information," *arXiv preprint arXiv:1503.03607*, 2015.

[77] H. Edelsbrunner, J. Harer, and A. K. Patel, "Reeb spaces of piecewise linear mappings," in *Proc. 24th Annu. Symp. Computational Geometry*. ACM, 2008, pp. 242–250.

[78] A. Chattopadhyay, H. Carr, D. Duke, and Z. Geng, "Extracting jacobi structures in reeb spaces," *EuroVis-Short Papers*, pp. 1–4, 2014.

[79] B. Strodthoff and B. Jüttler, "Layered reeb graphs for three-dimensional manifolds in boundary representation," *Computers and Graphics*, vol. 46, pp. 186–197, 2015.

[80] H. Carr and D. Duke, "Joint contour nets," *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 8, pp. 1100–1113, 2014.

[81] J. Tierny and H. Carr, "Jacobi fiber surfaces for bivariate reeb space computation," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 960–969, 2017.

[82] H. Lal, M. Rai, S. Karan, A. Verma, and D. Ram, "Multivariate hierarchical clustering of cowpea germplasm (vigna unguiculata (l.) walp." in *Int. Conf. Indigenous Vegetables and Legumes. Prospectus for Fighting Poverty, Hunger and Malnutrition 752*, 2006, pp. 413–416.

[83] Y. Gu and C. Wang, "A study of hierarchical correlation clustering for scientific volume data," in *Int. Symp. Visual Comput.* Springer, 2010, pp. 437–446.

[84] F. Fouedjio, "A hierarchical clustering method for multivariate geostatistical data," *Spatial Statist.*, vol. 18, pp. 333–351, 2016.

[85] R. Maronna and P. M. Jacovkis, "Multivariate clustering procedures with variable metrics," *J. Biometrics*, pp. 499–505, 1974.

[86] G. Luepke, J. M. Botbol, M. W. Lee, T. D. Light, D. R. Hutchinson, E. C. Escowitz, R. Tripp, and H. D. King, *Multivariate Clustering Based on Entropy*. US Government Printing Office, 1983, no. 1893-1897.

[87] T. Eliassi-Rad and T. Critchlow, "Multivariate clustering of large-scale scientific simulation data," Lawrence Livermore National Lab., CA (US), Tech. Rep., 2003.

[88] S. Chevallier, Q. Barthélemy, and J. Atif, "Metrics for multivariate dictionaries," *arXiv preprint arXiv:1302.4242*, 2013.

[89] J. Chen, D. Silver, and L. Jiang, "The feature tree: Visualizing feature tracking in distributed amr datasets," in *Proc. 2003 IEEE Symp. Parallel and Large-Data Visualization and Graphics*. IEEE Computer Society, 2003, p. 14.

[90] D. S. Kalivas and A. A. Sawchuk, "A region matching motion estimation algorithm," *CVGIP: Image Understanding*, vol. 54, no. 2, pp. 275–288, 1991.

[91] I. K. Sethi, N. V. Patel, and J. H. Yoo, "A general approach for token correspondence," *J. Pattern Recog.*, vol. 27, no. 12, pp. 1775–1786, 1994.

[92] B. K. Horn and B. G. Schunck, "Determining optical flow," in *1981 Tech. Symp. East*. International Society for Optics and Photonics, 1981, pp. 319–331.

[93] H.-H. Nagel, "On the estimation of optical flow: Relations between different approaches and some new results," *J. Artificial Intelligence*, vol. 33, no. 3, pp. 299–324, 1987.

[94] J. Aggarwal and N. Nandhakumar, "On the computation of motion from sequences of images-a review," DTIC Document, Tech. Rep., 1988.

[95] B. K. Horn and B. G. Schunck, "Determining optical flow: A retrospective," *J. Artificial Intelligence*, vol. 59, no. 1, pp. 81–87, 1993.

[96] J. Shi and C. Tomasi, "Good features to track," in *1994 IEEE Comp. Soc. Conf. Comput. Vis. and Pattern Recog. (CVPR), 1994.* IEEE, 1994, pp. 593–600.

[97] L. Torresani, V. Kolmogorov, and C. Rother, "Feature correspondence via graph matching: Models and global optimization," in *European Conf. Comput. Vision*. Springer, 2008, pp. 596–609.

[98] D. Ring and A. Kokaram, "User-assisted feature correspondence matching," in *Conf. Visual Media Production (CVMP), 2009.* IEEE, 2009, pp. 214–219.

[99] F. Zhou and F. De la Torre, "Factorized graph matching," in *2012 IEEE Conf. Comput. Vis. and Pattern Recog. (CVPR)*. IEEE, 2012, pp. 127–134.

[100] X. Yang, H. Qiao, and Z.-Y. Liu, "Feature correspondence based on directed structural model matching," *Image and Vision Comput.*, vol. 33, pp. 57–67, 2015.

[101] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1719–1749, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2011.01898.x

[102] Y. Frishman and A. Tal, "Online dynamic graph drawing," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 4, pp. 727–740, 2008.

[103] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "The state of the art in visualizing dynamic graphs," *EuroVis STAR*, 2014.

[104] C. Vehlow, F. Beck, P. Auwärter, and D. Weiskopf, "Visualizing the evolution of communities in dynamic graphs," in *Computer Graphics Forum*, vol. 34, no. 1. Wiley Online Library, 2015, pp. 277–288.

[105] J. Reynolds, "A hierarchical layout algorithm for drawing directed graphs," PhD dissertation, Queen's University, 1998.

[106] R. Tamassia, *Handbook of Graph Drawing and Visualization*. CRC press, 2013.

[107] J. N. Warfield, "Crossing theory and hierarchy mapping," *IEEE Trans. Syst., Man, Cybern.*, vol. 7, no. 7, pp. 505–523, 1977.

[108] M.-J. Carpano, "Automatic display of hierarchized graphs for computer-aided decision analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. 10, no. 11, pp. 705–715, 1980.

[109] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, no. 2, pp. 109–125, 1981.

[110] M. Fröhlich and M. Werner, "Demonstration of the interactive graph visualization system da vinci," in *Int. Symp. Graph Drawing*. Springer, 1994, pp. 266–269.

[111] M. Himsolt, "The graphlet system (system demonstration)," in *Int. Symp. Graph Drawing*. Springer, 1996, pp. 233–240.

[112] P. Eades and N. C. Wormald, "Edge crossings in drawings of bipartite graphs," *Algorithmica*, vol. 11, pp. 379–403, 1994, 10.1007/BF01187020. [Online]. Available: http://dx.doi.org/10.1007/BF01187020

[113] P. Eades and D. Kelly, "Heuristics for reducing crossings in 2-layered networks," *Ars Combin.*, vol. 21, pp. 89–98, 1986.

[114] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. 1993 IEEE/ACM Int. Conf. Computer-aided Design*, ser. ICCAD '93. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993, pp. 42–47. [Online]. Available: http://dl.acm.org/citation.cfm?id=259794.259802

[115] R. F. Cohen, G. Di Battista, R. Tamassia, I. G. Tollis, and P. Bertolazzi, "A framework for dynamic graph drawing," in *Proc. 8th Annu. Symp. Computational Geometry*. ACM, 1992, pp. 261–270.

[116] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *J. Visual Languages and Comput.*, vol. 6, no. 2, pp. 183–210, 1995.

[117] M. L. Huang, P. Eades, and J. Wang, "On-line animated visualization of huge graphs using a modified spring algorithm," *J. Visual Languages and Comput.*, vol. 9, no. 6, pp. 623–645, 1998.

[118] Y. Frishman and A. Tal, "Dynamic drawing of clustered graphs," in *Proc. IEEE Symp. Information Visualization (INFOVIS), 2004.* IEEE, 2004, pp. 191–198.

[119] M. Baur and T. Schank, *Dynamic Graph Drawing in Visone*. Univ., Fak. für Informatik, 2008.

[120] U. Brandes and S. R. Corman, "Visual unrolling of network evolution and the analysis of dynamic discourse," *Proc. IEEE Symp. Information Visualization (IV)*, vol. 2, no. 1, pp. 40–50, 2003.

[121] M. Burch and S. Diehl, "Timeradartrees: Visualizing dynamic compound digraphs," in *Computer Graphics Forum*, vol. 27, no. 3. Wiley Online Library, 2008, pp. 823–830.

[122] F. Reitz, "A framework for an ego-centered and time-aware visualization of relations in arbitrary data repositories," *arXiv preprint arXiv:1009.5183*, 2010.

[123] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf, "Parallel edge splatting for scalable dynamic graph visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2344–2353, 2011.

[124] M. Burch, B. Schmidt, and D. Weiskopf, "A matrix-based visualization for exploring dynamic compound digraphs," in *Proc. 2013 17th Int. Conf. Information Visualisation (IV)*. IEEE, 2013, pp. 66–73.

[125] W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci, "Interactive exploration of large-scale time-varying data using dynamic tracking graphs," in *2012 IEEE Symp. Large Data Anal. and Visualization (LDAV)*, Oct 2012, pp. 9–17.

[126] S. Hadlak, H.-J. Schulz, and H. Schumann, "In situ exploration of large dynamic networks," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2334–2343, 2011.

[127] A. Sallaberry, C. Muelder, and K.-L. Ma, "Clustering, visualizing, and navigating for large dynamic graphs," in *20th Int. Symp. Graph Drawing*, vol. 7704. Springer-Verlag Berlin Heidelberg, 2012, pp. 487–498.

[128] H. Yu, K.-L. Ma, and J. Welling, "I/o strategies for parallel rendering of large time-varying volume data." in *Eurographics/ACM SIGGRAPH Symp. Parallel Graphics and Visualization*, 2004, pp. 31–40.

[129] J. Wei, H. Yu, J. Chen, and K.-L. Ma, "Parallel clustering for visualizing large scientific line data," in *IEEE Symp. Large Data Anal. and Visualization, 2011*, Oct 2011, pp. 47–55.

[130] P. Sutton and C. Hansen, "Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon)," in *Proc. Conf. Visualization'99*, Oct 1999, pp. 147–520.

[131] A. Joshi, "Art-inspired techniques for visualizing time-varying data," Ph.D. dissertation, University of Maryland, Baltimore County, 2007.

[132] E. B. Lum, K.-L. Ma, and J. Clyne, "A hardware-assisted scalable solution for interactive volume rendering of time-varying data," *IEEE Trans. Vis. Comput. Graphics*, vol. 8, no. 3, pp. 286–301, 2002.

[133] A. Lu and H.-W. Shen, "Interactive storyboard for overall time-varying data visualization," in *Proc. 2008 IEEE Pacific Visualization Symp. (PacificVis)*. IEEE, 2008, pp. 143–150.

[134] C. Wang, H. Yu, and K.-L. Ma, "Importance-driven time-varying data visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 6, pp. 1547–1554, 2008.

[135] Y. Gu and C. Wang, "itree: Exploring time-varying data using indexable tree," in *Proc. 2013 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2013, pp. 137–144.

[136] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller, "Twitinfo: Aggregating and visualizing microblogs for event exploration," in *Proc. SIGCHI Conf. Human Factors in Comput. Systems*, ser. CHI '11. ACM, 2011, pp. 227–236.

[137] W. Dou, X. Wang, D. Skau, W. Ribarsky, and M. X. Zhou, "Leadline: Interactive visual analysis of text data through event identification and exploration." in *IEEE VAST*. IEEE Computer Society, 2012, pp. 93–102.

[138] R. Chang, M. Ghoniem, R. Kosara, W. Ribarsky, J. Yang, E. A. Suma, C. Ziemkiewicz, D. A. Kern, and A. Sudjianto, "Wirevis: Visualization of categorical, time-varying data from financial transactions," in *IEEE VAST*. IEEE, 2007, pp. 155–162.

[139] Z. Shen, J. Wei, N. Sundaresan, and K.-L. Ma, "Visual analysis of massive web session data," in *2012 IEEE Symp. Large Data Anal. and Visualization (LDAV)*. IEEE, 2012, pp. 65–72.

[140] T.-M. Rhyne, "Does the difference between information and scientific visualization really matter?" *IEEE Comput. Graph. Appl.*, vol. 23, no. 3, pp. 6–8, 2003.

[141] T.-M. Rhyne, M. Tory, T. Munzner, M. O. Ward, C. Johnson, and D. H. Laidlaw, "Information and scientific visualization: Separate but equal or happy together at last," in *Proc. IEEE Conf. Visualization'03*. IEEE Computer Society Press, 2003, pp. 619–621.

[142] H. Hauser, D. Weiskopf, K.-L. Ma, J. van Wijk, and R. Kosara, "Scivis, infovis - bridging the community divide?!" in *Proc. IEEE Conf. Visualization (Panel Proceedings)*. IEEE Computer Society Press, 2006, pp. 52–55.

[143] H. R. Nagel, "Scientific visualization versus information visualization," in *Workshop on State-of-the-art in Scientific and Parallel Comput., Sweden*. Citeseer, 2006.

[144] A. Kerren, J. T. Stasko, J.-D. Fekete, and C. North, "Workshop report: Information visualization-human-centered issues in visual representation, interaction, and evaluation." *Proc. IEEE Symp. Information Visualization (IV)*, vol. 6, no. 3, pp. 189–196, 2007.

[145] D. L. Gresh, B. E. Rogowitz, R. L. Winslow, D. F. Scollan, and C. K. Yung, "Weave: A system for visually linking 3-d and statistical visualizations applied to cardiac simulation and measurement data," in *Proc. Conf. Visualization'00*, Oct 2000, pp. 489–492.

[146] R. Kosara, F. Bendix, and H. Hauser, "Time histograms for large, time-dependent data," in *Proc. 6th Joint Eurographics - IEEE TCVG Conf. Visualization*. Eurographics Association, 2004, pp. 45–54.

[147] G. Reina and T. Ertl, "Volume visualization and visual queries for large high-dimensional datasets," in *Proc. 6th Joint Eurographics - IEEE TCVG Conference on Visualization*. Eurographics Association, 2004, pp. 255–260.

[148] M. Tory, S. Potts, T. Möller, and I. Member, "A parallel coordinates style interface for exploratory volume visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 11, p. 2005, 2005.

[149] M. Tory and T. Möller, "A model-based visualization taxonomy," *Computing Science Department, Simon Fraser University, Technical Report No. TR*, vol. 6, 2002.

[150] H. Piringer, R. Kosara, and H. Hauser, "Interactive focus+context visualization with linked 2d/3d scatterplots," in *Proc. 2nd Int. Conf. Coordinated and Multiple Views in Exploratory Visualization*. IEEE Computer Society Press, 2004, pp. 49–60.

[151] H. Hauser, "Toward new grounds in visualization," *SIGGRAPH Computer Graphics*, vol. 39, no. 2, pp. 5–8, May 2005.

[152] M. Jern and J. Franzen, "Integrating infovis and geovis components," in *Proc. 11th Int. Conf. Information Visualization (IV), 2007.*, July 2007, pp. 511–520.

[153] B. Wylie and J. Baumes, "A unified toolkit for information and scientific visualization," in *IS&T/SPIE Electronic Imaging 2009, Visual Data Analytics (VDA 2009)*, vol. 7243. International Society for Optics and Photonics, 2009, pp. 72 430H–72 430H–16.

[154] D. F. Keefe, M. Ewert, W. Ribarsky, and R. Chang, "Interactive coordinated multiple-view visualization of biomechanical motion data," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 1383–1390, 2009.

[155] Wikipedia, "Maslow's hierarchy of needs," 2002, [Online; accessed 12-March-2017]. [Online]. Available: https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs

[156] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Reading, MA, 1990, vol. 199.

[157] J. Navarro and S. D. White, "The structure of cold dark matter halos," in *Symp. Int. Astronomic. Union*, vol. 171. Kluwer Academic Publishers Group, 1996, pp. 255–258.

[158] S. Ghigna, B. Moore, F. Governato, G. Lake, T. Quinn, and J. Stadel, "Dark matter haloes within clusters," *Monthly Notices Roy. Astro. Soc.*, vol. 300, no. 1, pp. 146–162, 1998.

[159] M. Davis, G. Efstathiou, C. S. Frenk, and S. D. White, "The evolution of large-scale structure in a universe dominated by cold dark matter," *Astrophys. J*, vol. 292, pp. 371–394, 1985.

[160] S. Gottlöber, A. Klypin, and A. V. Kravtsov, "Merging history as a function of halo environment," *Astrophys. J*, vol. 546, no. 1, p. 223, 2001.

[161] M. White, L. Hernquist, and V. Springel, "The halo model and numerical simulations," *Astrophys. J. Lett.*, vol. 550, no. 2, p. L129, 2001.

[162] W. Widanagamaachchi, P.-T. Bremer, C. Sewell, L.-T. Lo, J. Ahrens, and V. Pascucci, "Data-parallel halo finding with variable linking lengths," in *2014 IEEE 4th Symp. Large Data Anal. and Visualization (LDAV)*, Nov 2014, pp. 27–34.

[163] L.-t. Lo, C. Sewell, and J. Ahrens, "Piston: A portable cross-platform framework for data-parallel visualization operators," in *Eurographics Symp. Parallel Graphics and Visualization*, 2012.

[164] A. Henderson, "Paraview guide, a parallel visualization application. kitware inc.(2005)," *URL http://www. paraview. org*, 2005.

[165] M. Day, J. Bell, P.-T. Bremer, V. Pascucci, V. Beckner, and M. Lijewski, "Turbulence effects on cellular burning structures in lean premixed hydrogen flames," *Proc. Combust. and Flame*, vol. 156, no. 5, pp. 1035 – 1045, 2009.

[166] K. Antypas, "Nersc-6 workload analysis and benchmark selection process," *Lawrence Berkeley National Laboratory*, 2008.

[167] W. Widanagamaachchi, P. Klacansky, H. Kolla, A. Bhagatwala, J. Chen, V. Pascucci, and P. T. Bremer, "Tracking features in embedded surfaces: Understanding extinction in turbulent combustion," in *2015 IEEE 5th Symp. Large Data Anal. and Visualization (LDAV)*, Oct 2015, pp. 9–16.

[168] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int. Conf. Computer Vision*, vol. 2, July 2001, pp. 416–423.

[169] T. Liu, C. Jones, M. Seyedhosseini, and T. Tasdizen, "A modular hierarchical approach to 3d electron microscopy image segmentation," *J. Neuroscience Methods*, vol. 226, pp. 88–102, 2014.

[170] T. Liu, M. Seyedhosseini, and T. Tasdizen, "Image segmentation using hierarchical merge tree," *IEEE Trans. Image Process.*, vol. 25, no. 10, pp. 4596–4607, 2016.

[171] J. M. Gauch and S. M. Pizer, "Multiresolution analysis of ridges and valleys in grey-scale images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 6, pp. 635–646, 1993.

[172] W. Widanagamaachchi, P.-T. Bremer, and V. Pascucci, "Combining nested feature hierarchies for bivariate feature exploration," in *Proc. 2017 Conf. Visualization*, 2017, (In submission).

[173] M. S. F. V. D. Pondeca, G. S. Manikin, G. DiMego, S. G. Benjamin, D. F. Parrish, R. J. Purser, W.-S. Wu, J. D. Horel, D. T. Myrick, Y. Lin, R. M. Aune, D. Keyser, B. Colman, G. Mann, and J. Vavra, "The real-time mesoscale analysis at NOAA's national centers for environmental prediction: Current status and development," *J. Weather and Forecasting*, vol. 26, no. 5, pp. 593–612, 2011.

[174] Y. Yang and M. H. Ritzwoller, "Teleseismic surface wave tomography in the western US using the transportable array component of USArray," *J. Geophys. Res. Lett.*, vol. 35, no. 4, 2008.

[175] G. L. Pavlis, K. Sigloch, S. Burdick, M. J. Fouch, and F. L. Vernon, "Unraveling the geometry of the farallon plate: Synthesis of three-dimensional imaging results from USArray," *J. Tectonophys.*, vol. 532, pp. 82–102, 2012.

[176] W. Widanagamaachchi, A. Jacques, B. Wang, E. Crosman, P.-T. Bremer, V. Pascucci, and J. Horel, "Exploring the evolution of pressure-perturbations to understand atmospheric phenomena," in *Proc. 2017 IEEE Pacific Visualization Symp. (PacificVis)*, Apr 2017.

[177] B. Zhang, J. Wang, M. Li, and Q. Hou, "A molecular dynamics study of helium bubble formation and gas release near titanium surfaces," *J. Nuclear Materials*, vol. 438, no. 1, pp. 178–182, 2013.

[178] F. Sefta, K. D. Hammond, N. Juslin, and B. D. Wirth, "Tungsten surface evolution by helium bubble nucleation, growth and rupture," *J. Nuclear Fusion*, vol. 53, no. 7, p. 073015, 2013.

[179] Y. Zhou, J. Wang, Q. Hou, and A. Deng, "Molecular dynamics simulations of the diffusion and coalescence of helium in tungsten," *J. Nuclear Materials*, vol. 446, no. 1, pp. 49–55, 2014.

[180] W. Widanagamaachchi, K. Hammond, L.-T. Lo, B. Wirth, F. Samsel, E. Bertini, J. Kennedy, and E. Puppo, "Visualization and analysis of large-scale atomistic simulations of plasma-surface interactions," in *Eurographics Conf. Visualization (EuroVis)-Short Papers*. The Eurographics Association, 2015.

[181] M. S. Day and J. B. Bell, "Numerical simulation of laminar reacting flows with complex chemistry," *Proc. Combust. Theory and Modelling*, vol. 4, no. 4, pp. 535–556, 2000.

[182] J. Jeong and F. Hussain, "On the identification of a vortex," *J. Fluid Mechanics*, vol. 285, pp. 69–94, Apr. 1995. [Online]. Available: http://www.journals.cambridge.org/abstract_S0022112095000462

[183] J. del Álamo, J. Jimenez, P. Zandonade, and R. D. Moser, "Self-similar vortex clusters in the turbulent logarithmic region," *J. Fluid Mechanics*, vol. 561, pp. 329–358, 2006.

[184] R. Smith and P. Gent, "Reference manual for the parallel ocean program (pop)," *Los Alamos Unclassified Report LA-UR-02-2484*, 2002.

[185] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, S. Philip, and S. Kumar, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Chapman & Hall/Crc Computational Science, 2012, ch. The ViSUS Visualization Framework.

[186] V. Pascucci, P.-T. Bremer, A. Gyulassy, G. Scorzelli, C. Christensen, B. Summa, and S. Kumar, *Advances in Parallel Computing, Cloud Computing and Big Data*. IOS Press, 2013, vol. 23, ch. Scalable Visualization and Interactive Analysis Using Massive Data Streams, pp. 212–230.

[187] J. Davies, *d3-cloud:Word Cloud Layout*, 2012, available at https://github.com/jasondavies/d3-cloud/.

[188] E. Hazzard, *Openlayers 2.10 Beginner's Guide*. Packt Publishing Ltd, 2011.

[189] Google Team, *Google Maps*, Google, 2005, available at http://www.google.com/maps/.

[190] Awesomium Team, *Awesomium:HTML UI Engine*, 2013, available at http://www.awesomium.com/.

[191] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphvizopen source graph drawing tools," in *Int. Symp. Graph Drawing*. Springer, 2001, pp. 483–484.

[192] A. Jacques, "Temporal and spatial analyses of pressure perturbations from the USArray transportable array," PhD dissertation, Dept. of Atmospheric Sciences, University of Utah, 2016.

[193] "USArray microbarograph gui interface," http://meso1.chpc.utah.edu/usarray/, accessed: 2016-09-26.

[194] A. Bhagatwala, Z. Luo, H. Shen, J. A. Sutton, T. Lu, and J. H. Chen, "Numerical and experimental investigation of turbulent dme jet flames," *Proc. Combust. Inst.*, vol. 35, no. 2, pp. 1157–1166, 2015.

[195] G. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci, "Feature tracking using reeb graphs," in *Topological Methods in Data Analysis and Visualization*, ser. Mathematics and Visualization, 2011, pp. 241–253.

[196] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L.-W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark, "Multiparameter intelligent monitoring in intensive care ii (mimic-ii): A public-access intensive care unit database," *J. Critical Care Medicine*, vol. 39, no. 5, p. 952, 2011.

[197] J. Lee, D. M. Maslove, and J. A. Dubin, "Personalized mortality prediction driven by electronic medical data and a patient similarity metric," *PloS One*, vol. 10, no. 5, p. e0127428, 2015.

[198] W. Widanagamaachchi, Y. Livnat, P.-T. Bremer, S. Duvall, and V. Pascucci, "Interactive visualization and exploration of patient progression in a hospital setting," 2016, Workshop on Visual Analytics in Healthcare.

[199] ——, "Interactive visualization and exploration of patient progression in a hospital setting," in *Proc. AMIA 2017 Annu. Symp.*, 2017 (To appear).